



**Advanced Cyber-Threat Intelligence, Detection, and Mitigation
Platform for a Trusted Internet of Things
Grant Agreement: 786698**

D6.2 CYBER-TRUST device tools

Work Package 6: Advanced cyber-attack detection and mitigation

Document Dissemination Level

| | | |
|----|--|-------------------------------------|
| P | Public | <input checked="" type="checkbox"/> |
| CO | Confidential, only for members of the Consortium (including the Commission Services) | <input type="checkbox"/> |

Document Due Date: 31/01/2020

Document Submission Date: 31/01/2020



Co-funded by the Horizon 2020 Framework Programme of the European Union



Document Information

| | |
|---------------------------------|---|
| Deliverable number: | 6.2 |
| Deliverable title: | CYBER-TRUST device tools |
| Deliverable version: | 1.0 |
| Work Package number: | 6 |
| Work Package title: | Advanced cyber-attack detection and mitigation |
| Due Date of delivery: | 31/01/2020 |
| Actual date of delivery: | 31/01/2020 |
| Dissemination level: | PU |
| Editor(s): | Michael Skitsas (ADITESS) |
| Contributor(s): | Elisavet Charalambous (ADITESS), George Boulougaris (ADITESS), Christos Katomoniatis (ADITESS) |
| Reviewer(s): | Raymond Binnendijk, Gohar Sargsyan (CGI), Stefano Cuomo (MATHEMA), Simone Naldini (MATHEMA) |
| Project name: | Advanced Cyber-Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things |
| Project Acronym | Cyber-Trust |
| Project starting date: | 1/5/2018 |
| Project duration: | 36 months |
| Rights: | Cyber-Trust Consortium |

Version History

| Version | Date | Beneficiary | Description |
|------------|------------|--------------|---|
| 0.1 | 04/12/2019 | ADITESS | Deliverable's ToC |
| 0.5 | 10/01/2020 | ADITESS | Sections added to master document |
| 0.6 | 17/01/2020 | ADITESS | Sections added to master document |
| 0.7 | 20/01/2020 | ADITESS | Finalize content of deliverable |
| 0.8 | 27/01/2020 | ADITESS | Formatting of document and release for review |
| 0.9 | 29/01/2020 | CGI, MATHEMA | Review received and applying changes |
| 1.0 | 30/01/2020 | ADITESS | Final version of deliverable |

Acronyms

| ACRONYM | EXPLANATION |
|---------|--|
| AAS | Authentication and Authorization Service |
| API | Application Programming Interface |
| CPE | Common Platform Enumeration |
| CRUD | Create, Retrieve, Update, Delete |
| CVE | Common Vulnerability Enumeration |
| DB | Database |
| DLT | Distributed ledger technology |
| DMS | Data Management System |
| eVDB | Enhanced Vulnerability Database) |
| IDS | Intrusion Detection System |
| IoT | Internet of Things |
| ISO | International Organization for Standardization |
| ISP | Internet Service Provider |
| JSON | JavaScript Object Notation |
| MISP | Malware Information Sharing Platform |
| OS | Operating System |
| PS | Profiling Service |
| REST | Representational State Transfer |
| SDA | Smart Device Agent |
| SDM | Smart Device Module |
| SGA | Smart Gateway Agent |
| SGM | Smart Gateway Module |
| SSL | Secure Socket Layers |

Table of Contents

| | |
|---|-----------|
| Table of Contents | 4 |
| Executive Summary | 6 |
| 1. Introduction | 7 |
| 1.1 Purpose of the document | 7 |
| 1.2 Relations to other activities in the project | 7 |
| 1.3 Structure of the document | 7 |
| 2. Device Profiling Service | 9 |
| 2.1 Profiling Service Interface | 12 |
| 2.2 Integration and synchronization with other data sources | 15 |
| 2.2.1 Synchronization with eVDB (A09) | 16 |
| 2.2.2 Patch DB | 17 |
| 2.2.2.1 Patch DB Crawling Service | 18 |
| 2.2.3 Data Correlation | 19 |
| 2.3 Vulnerabilities, Compromised Devices, Attacks and Mitigations | 21 |
| 2.4 Monitoring Rules, Alerts and Notifications | 22 |
| 2.4.1 Rules and Alerts | 23 |
| 2.4.2 Notification messages | 24 |
| 2.4.2.1 Component to Component Notifications | 24 |
| 2.4.2.2 Component to End-User Notifications | 25 |
| 3. IoT Device Monitoring | 26 |
| 3.1 Android-Based OS Devices (Smartphones, Smart TVs) | 27 |
| 3.1.1 Performance Monitoring | 27 |
| 3.1.2 Critical OS and Firmware Integrity Monitoring | 28 |
| 3.1.3 Device Level Network Monitoring | 30 |
| 3.2 Devices running a Linux-/Windows-based OS distribution | 32 |
| 3.2.1 Performance Monitoring | 32 |
| 3.2.2 Critical OS and Firmware Integrity Monitoring | 33 |
| 3.2.3 Device Level Network Monitoring | 33 |
| 3.3 Devices implemented to use IoT cloud Services | 34 |
| 3.3.1 Create new device | 34 |
| 3.3.2 IoT Cloud Services Data | 35 |
| 3.3.3 IoT Cloud Services Integration | 36 |
| 4. Device Forensic Evidence Collection | 39 |
| 5. Conclusion | 41 |

Table of Figures

| | |
|---|----|
| Figure 1 Profiling Service (A17) Conceptual Diagram..... | 9 |
| Figure 2 Profiling Service browsable REST API | 12 |
| Figure 3 Example of event processing during eVDB (A09) sync | 17 |
| Figure 4 Example of Samsung Firmware Repository | 18 |
| Figure 5 eVDB ISS Workflow in Profiling Service | 20 |
| Figure 6 Patch DB Crawler - Correlator Workflow..... | 20 |
| Figure 7 Notifications page on Cyber-Trust Mobile APP | 25 |
| Figure 8 High Level diagram of Monitor Activity | 26 |
| Figure 9 Device Status Page (Performance Monitoring) | 28 |
| Figure 10 Example of output from netstat -pln in Linux | 34 |
| Figure 11 Google Cloud Platform IoT Web Console | 35 |
| Figure 12 Pub/Sub Platform Webconsole | 37 |
| Figure 13 Data Flow for Evidence DB | 40 |

Table of Tables

| | |
|---|----|
| Table 1 Profiling Service Entities/Objects..... | 10 |
| Table 2 Core DMS system fields | 11 |
| Table 3 Profiling Service REST API End-points Summary..... | 12 |
| Table 4 User fields in Profiling Service | 14 |
| Table 5 Device fields in Profiling Service | 14 |
| Table 6 Smarthome fields in Profiling Service | 15 |
| Table 7 Monitoring Data message structure | 15 |
| Table 8 Profiling Service end-points for eVDB (A09) | 16 |
| Table 9 Patch DB REST API end-points | 19 |
| Table 10 Patch DB fields | 19 |
| Table 11 Vulnerability fields in Profiling Service | 21 |
| Table 12 Vulnerabilities REST API end-points..... | 21 |
| Table 13 Compromised Devices fields in Profiling Service | 21 |
| Table 14 Attack fields in Profiling Service..... | 22 |
| Table 15 Mitigation fields in Profiling Service | 22 |
| Table 16 Rule fields in Profiling Service..... | 23 |
| Table 17 Alert fields in Profiling Service | 23 |
| Table 18 Performance Monitoring fields for Devices..... | 27 |
| Table 19 Critical OS Files Integrity Check fields for Devices | 29 |
| Table 20 Status and About fields for Devices | 29 |
| Table 21 Installed Application fields for Devices..... | 30 |
| Table 22 LSOF command fields for Devices..... | 30 |
| Table 23 Mobile App netstat fields | 31 |
| Table 24 Mobile App ifconfig fields | 31 |
| Table 25 Evidence DB Metadata Fields | 39 |

Executive Summary

This report is a contractual deliverable within the Horizon 2020 Project Cyber-Trust: Advanced Cyber-Threat Intelligence, Detection, and Mitigation Platform for a Trusted Internet of Things. It provides detailed description of the work done under the Task 6.2 “Device tampering detection and remediation” related with the device monitoring tools. Apart from the technical documentation of tools that are developed and deployed on devices, the document presents the section of Profiling Service (A17) related with the monitoring of devices.

Considering the three main classes of IoT devices based on the hosted Operating System (OS) as introduced in D6.1 “State-of-the-art on profiling, detection and mitigation”, several tools and services, Data Aggregation Services, have been implemented for acquiring data and metrics related with the performance, network and binary files of the IoT device. Pre-defined rules are applied for the detection of abnormal and suspicious activity at device level. To support this activity, a detailed REST API documentation about the Profiling Service is presented which is responsible for the monitoring data management and correlation of devices with threat intelligence and available updates.

Attacks and mitigation actions, as a result of the monitoring activity, will be reported in the second deliverable of this task, D6.6 “Device-level attacks: proposed solutions”.

1. Introduction

Cyber-Trust project aims to combat potential threats posed by the adoption of IoT by building a proactive cyber-threat intelligence gathering and sharing platform. As the services being offered via IoT platforms are becoming highly pervasive, ubiquitous, and distributed, any concerns about our society's security are amplified due to the appearance of new forms of sophisticated threats and cyber-attacks. However, IoT devices are essentially resource-constrained in terms of computation, battery power, intermittent connectivity, and network protocols. In which, achieving the Cyber-Trust ultimate goal requires investigating and developing optimal solutions compatible with such limitations and constraints that exist in IoT.

1.1 Purpose of the document

The main objective of this deliverable is to provide a technical documentation of the tools and services that have been developed to enable the monitoring of, and integration with, IoT devices within the Cyber-Trust platform. In particular, the content of the deliverable includes details of the A03m, Smart Device Module (SDM) as well as the A17, Device Profiling Service (PS) which is highly related with the monitoring activity.

The responsible component for monitoring the end-user IoT device and establish the links with the Cyber-Trust core components hosted on the service provider layer is the SDM. The SDM component is primarily for the monitoring of device's usage, critical files, security status (patching status, firmware integrity, vulnerability risk) as well as suspicious network transactions, and secondly for the application of mitigation policies and remediation actions after the detection of an attack or threat that could endanger the integrity and operation of the monitored device. Due to its intended operation, the SDM is designed to check whether the hosting device performs as intended by its manufacturer, ensures that critical OS files are uncompromised and that only secure means of communication are used. Furthermore, SDM component is responsible for the application of mitigation policies and remediation actions after the detection of an attack or threat that could endanger the integrity and operation of the monitored device.

In case of detection of suspicious traffic and activity, network packages are signed by SDM and communicated with the Cyber-Trust Defense Service for further investigation. Data from the SDM are communicated to the Profiling Service (A17), responsible for the storage and management of Cyber-Trust generated and acquired data. In particular, two separate access control layers are supported: one for defining architectural policies and one for controlling runtime operations for matching use preferences.

1.2 Relations to other activities in the project

The deliverable D6.2 is one of the two deliverables linked with Task 6.2 "Device tampering detection and remediation" and implements the practices suggested in D6.1 "State-of-the-art on profiling, detection and mitigation" related with Cyber-Trust device tools and monitoring activities. Additionally, the integration with other sources (part of the Profiling Service, A17) is linked with the eVDB (A07, A09) component where the correlation with devices is done. This deliverable is also linked with WP7, Distributed ledger technology (DLT)(A02) and the Trust Management System (TMS)(A05) where notifications about suspicious activity of devices is communicated.

1.3 Structure of the document

The rest of the document is comprised of the following four (4) sections:

- Section 2: Device Profiling Service where architectural details about the Profiling Service as well as the documentation of provided REST API is given.
- Section 3: IoT Device Monitoring where the developed and deployed tools for performance, critical OS and network monitoring at device level is implemented.

- Section 4: Device Forensic Evidence Collection where the Evidence DB for devices is described.
- Section 5: Conclusion

2. Device Profiling Service

This component has a twofold responsibility: the central storage of device profiles and the correlation of existing information with newly acquired data from other repositories and sources (such as Patch DB for patch and firmware updates as well as manufacturer use documentation). The profiling service (A17) is the primary interface between the SDM and the Cyber-Trust backend components, and the responsible component for gathering and collecting information from the deployed SDAs (Smart Device Agents) and SGAs (Smart Gateway Agents). The main objective of this section is to provide the technical details of the Profiling Service that are necessary for the implementation and completion of monitoring activity by the Cyber-Trust related device tools. In particular, a high-level presentation of the Profiling Service architect, the data messages and the end-points as well as the correlation between devices and vulnerabilities are the topics that explained in the rest of the section.

Figure 1 illustrates the conceptual view of the Data Management System and Profiling Service. The architecture follows a loosely coupled approach allowing for scalability and extensibility where necessary. Direct interfacing with internal components and repositories is not supported due to security reasons. The Profiling Service ensures data integrity by maintaining a data provisioning system allowing tracking and monitoring of data evolution.

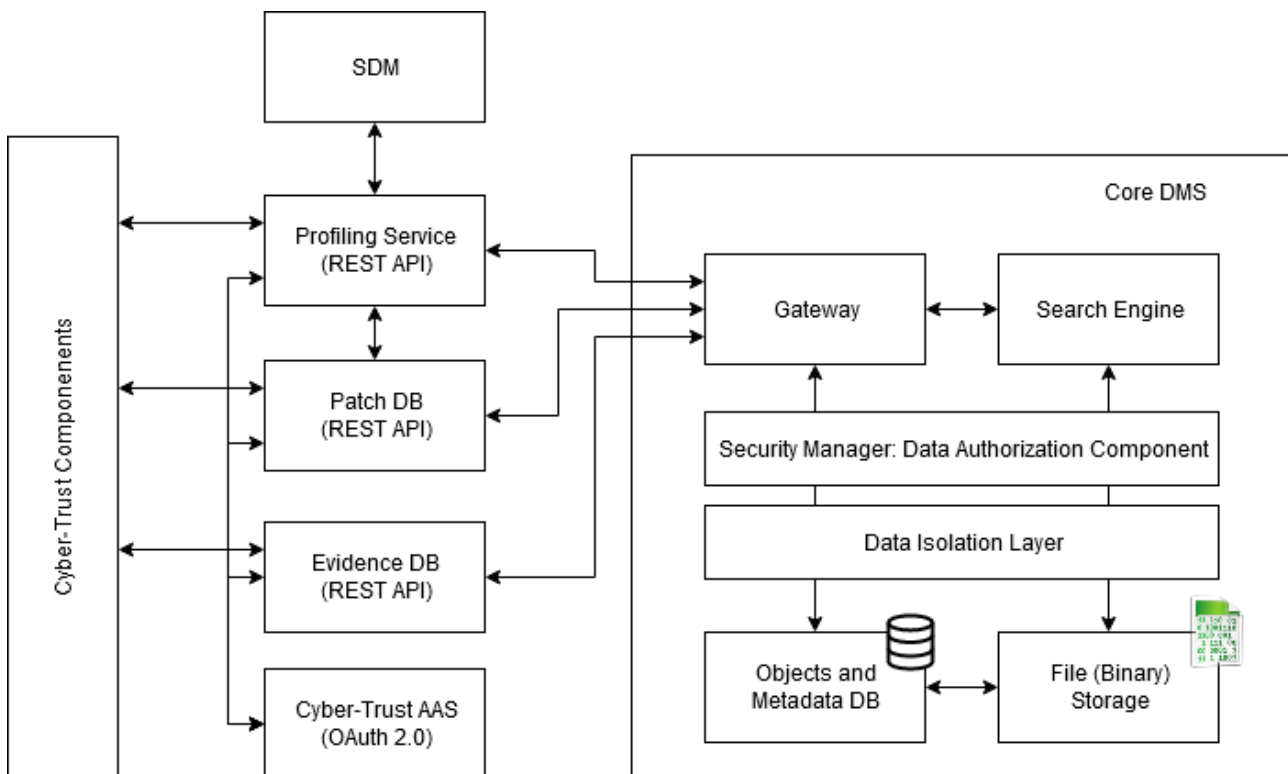


Figure 1 Profiling Service (A17) Conceptual Diagram

The core component of the profiling service, core DMS, is the data management system which is based on ADITESS VMCMS-GE, the Versatile Media Content Management System – Generic Edition. VMCMS-GE is a content management system for the efficient storage, management archiving, processing and logging of multimedia/heterogeneous content through a modular architecture. DMS enables data storage (CRUD operations) with search capabilities. The core DMS system does not provide direct access to any component or user of the system. The access is granted only through adaptors (such as Profiling Service) that exposed REST API. In the case of Cyber-Trust, three adaptors have been prepared to provide different interfaces of

data repositories over a common (shared) storage infrastructure ensuring the data isolation between them: (a) Profiling Service, (b) Patch DB and, (c) Evidence DB.

Each adaptor is consuming the API provided by the Gateway component and a registration phase should be followed. The registration phase of Adaptors to the DMS is internal process and is done by the developer. The authentication/authorization for each adaptor is done by the Security Manager component.

In order to isolate data between different users (i.e. user of Profiling Service, user of Patch DB, etc.) of DMS system, the data isolation layer is placed between the data storage engines and the gateway. With this layer, we maintained to provide logical isolated data (physical storage is the same) without any interaction between them. With minor modifications, physical isolation can be supported as well. In Cyber-Trust project and Profiling service we have three isolated data categories, one for each adaptor. This means that the interaction between the three components can be done only through the available REST API.

Accessing data of profiling service through the REST API (for all cases) is secured by the Cyber-Trust Authentication and Authorization Service (ASS) implementing OAuth2 protocol. Through the AAS, the following features are supported by the profiling service for the Cyber-Trust users.

- Data access rights (based on the authentication token the Profiling Service will grant access only to relevant services)
- Certificate negotiation is performed through the AAS service and is based on OAuth2.0 protocol

Additionally, the communication between both internal and external components is routed through secure connections.

In the DMS, the data integrity of the original information is preserved with the computation of relevant hashes and digital signatures for any embedded multimedia object. The original data along with integrity preserving metrics are stored in a separate repository in which no user may perform changes. Additionally, and based on the imposed policy, data ageing attributes are attached to the object ensuring data retention only for the necessary and predefined period of time. The renewal of an item's date of expiration is possible and needs to be initiated from a user with the appropriate access key (separated from other functional permissions). All incoming and outgoing actions/requests are logged and retained to the system in the form of a record; only visible to users with appropriate permissions.

The Profiling Service is responsible for the data and metadata management of the following entities:

Table 1 Profiling Service Entities/Objects

| Entity / Object | Description |
|------------------------|--|
| Users | User entity is responsible to keep all data related with the registered user that are required by the Profiling Service (device profile) for notification and other activities. |
| ISP | Information related with ISP that the profiling service instance belongs. |
| Devices | Data related with devices. For each device of the system, a record with information is stored in the profiling service under Devices entity. Also, enriched data as a result of correlations are also stored under the device information. |
| Smarthomes | Data related with smarthomes. Identification details and links to connected devices and users will be kept under this Entity. |
| Vulnerabilities | Vulnerabilities entity is responsible to store information with vulnerabilities (as identified by Cyber-Trust) related with registered devices. Currently, vulnerabilities are synced with the eVDB (A09) and the information is kept in four objects. |
| Mitigations | The mitigations proposed by Cyber-Trust as well as the actions and the status will be stored in the Profiling Service. Mitigations will be correlated (linked) with devices. |

| | |
|----------------------------|---|
| Compromised Devices | When a device is compromised, appropriate record(s) will be created and stored in the Profiling Service. |
| Attacks | All the detected attacks will be stored and linked with devices in the Profiling Service. Status and additional info will be included in the attack's entity. |
| Rules | The list of rules that will be applied on registered devices and monitoring date. Rules are prepared by the Cyber-Trust team and uploaded to profiling service for storage and retrieval. |
| Alerts | All the generated alerts by the Cyber-Trust devices are logged in the alert's entity. |
| Notifications | Similar with alerts, all the notifications that sent to end-users are kept in the notification's entity. |
| Monitoring Data | Profiling service keep track of all monitoring data. The retention policy of tracked data can be define for each device type, data type, etc. The monitoring data of the profiling service is the source of Evidence DB data. |

Additional to the object and metadata management, Profiling Service provides functionalities for the management of binary files whilst also allowing for their downloading over a temporary URL.

Each record that it is stored in the Profiling Service (regardless the service) is enhanced with a set of system fields prepared by core DMS. System fields are related with privacy and security aspects and are automatically calculated by the system and a prefix of “_” exists (i.e. _timeToArchive). Table 2 presents the list of available fields within Cyber-Trust project.

Table 2 Core DMS system fields

| Field | Type | Description |
|---------------------------|-------------------|---|
| md5Checksum | String | The calculated MD5 string for the object |
| binaryID | String (UUID Ref) | Reference to an associate binary of the current object |
| timeToDelete | Date (ISO 8601) | Time to delete the object based on retention policy. |
| timeToArchive | Date (ISO 8601) | Time to archive the object based on retention police. Time to archive is less or equal with the time to delete field. |
| id | String (Unique) | Unique ID of the object in the system. Additional ID (object ID) may exists and is controlled by the user. |
| type | String | Internal type of the object. This fields indicates the collection (entity) of the object. |
| instertedTimestamp | Date (ISO 8601) | Automatically calculated field when a record is created |
| updatedTimestamp | Date (ISO 8601) | Automatically calculated field when a record is updated |

2.1 Profiling Service Interface

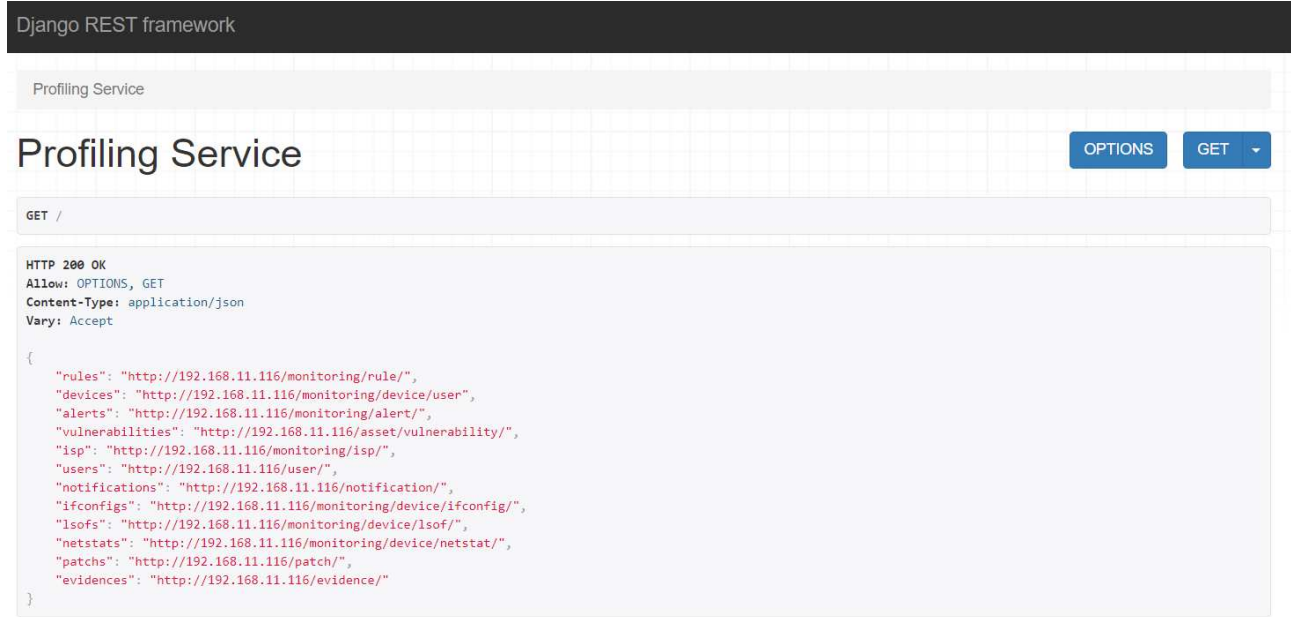


Figure 2 Profiling Service browsable REST API

The Profiling Service interface component is responsible for the reception and forwarding of messages from and to Cyber-Trust components and devices, through a public Rest API. Upon the reception of an object, the interface component is responsible for discriminating its parts into primitive data objects (the result is a list of metadata objects, binary data or a combination).

Figure 2 presents an example of the browsable REST API while Table 3 provides an overview of the implemented end-points withing Profiling Service. Based on the available method, each point can provide different type of functionality. However, beyond the basic CRUD operations, Profiling Service is integrated with correlation and notification engines.

Table 3 Profiling Service REST API End-points Summary

| End-point | Method | Data | Description |
|------------------------|--------|---------|---|
| /user | GET | | Retrieve the list with available users. |
| /user | POST | Table 4 | Insert new user to the profiling service |
| /user/<id> | GET | | Details of the user with given <id> |
| /user/<id> | PATCH | | Update details of the user with <id> |
| /user/<id> | DELETE | | Delete the user with <id> |
| /device | GET | | Retrieve the list with available devices. |
| /device | POST | | Insert new device to the profiling service |
| /device/<id> | GET | | Details of the user with given <id> |
| /device/<id> | PATCH | | Update details of the user with <id> |
| /device/<id> | DELETE | | Delete the user with <id> |
| /device/user/<id> | GET | | Retrieve all devices of user <id> |
| /device/smarthome/<id> | GET | | Retrieve all registered devices to smarthome <id> |
| /smarthome | GET | | Retrieve the list with available smarthomes. |
| /smarthome | POST | | Insert new smarthome to the profiling service |
| /smarthome/<id> | GET | | Details of the smarthome with given <id> |
| /smarthome/<id> | PATCH | | Update details of the smarthome with <id> |

| | | | |
|-------------------------------|--------|--|--|
| /smarthome/<id> | DELETE | | Delete the smarthome with <id> |
| /monitoring/<id> | POST | | Create new monitoring instance of fields in the profiling service for device <id> |
| /monitoring/performance/<id> | POST | | Retrieve performance monitoring data for the device <id>. Filtering options are allowed to the body of POST message |
| /monitoring/files/<id> | POST | | Retrieve performance monitoring data for the device <id>. Filtering options are allowed to the body of POST message |
| /monitoring/systeminfo/<id> | POST | | Retrieve system information data that are monitored for the device <id>. Filtering options are allowed to the body of POST message |
| /monitoring/applications/<id> | POST | | Retrieve applications monitoring data for the device <id>. Filtering options are allowed to the body of POST message |
| /monitoring/lsof/<id> | POST | | Retrieve lsof command related fields that are monitored for the device <id>. Filtering options are allowed to the body of POST message |
| /monitoring/netstat/<id> | POST | | Retrieve netstat related monitoring data for the device <id>. Filtering options are allowed to the body of POST message |
| /monitoring/ifconfig/<id> | POST | | Retrieve ifconfig related monitoring data for the device <id>. Filtering options are allowed to the body of POST message |
| /rule | GET | | Retrieve the list with available devices. |
| /rule | POST | | Insert new device to the profiling service |
| /rule/<id> | GET | | Details of the user with given <id> |
| /rule/<id> | PATCH | | Update details of the user with <id> |
| /rule/<id> | DELETE | | Delete the user with <id> |
| /rule/type/<type> | GET | | Retrieve available rules for devices of type <type> |
| /rule/device/<id> | GET | | Retrieve all rules for the device <id> |
| /alert | GET | | Retrieve the list with available devices. |
| /alert | POST | | Insert new device to the profiling service |
| /alert/<id> | GET | | Details of the user with given <id> |
| /alert/<id> | PATCH | | Update details of the user with <id> |
| /alert/<id> | DELETE | | Delete the user with <id> |
| /alert/device/<id> | GET | | Get the list of device alerts |
| /notification | GET | | Retrieve the list with available devices. |
| /notification | POST | | Insert new device to the profiling service |
| /notification/<id> | GET | | Details of the user with given <id> |
| /notification/<id> | PATCH | | Update details of the user with <id> |
| /notification/<id> | DELETE | | Delete the user with <id> |
| /mitigation | GET | | Retrieve the list with recorded mitigations. |
| /mitigation | POST | | Insert new mitigation to the profiling service |
| /mitigation/<id> | GET | | Details of the mitigation with given <id> |
| /mitigation/<id> | PATCH | | Update details of the mitigation with <id> |
| /mitigation/<id> | DELETE | | Delete the mitigation with <id> |

| | | | |
|----------------------------|--------|--|--|
| /mitigation/device/<id> | GET | | Retrieve the list with mitigations for specific device <id>. |
| /attack | GET | | Retrieve the list with recorded attacks. |
| /attack | POST | | Insert new attack to the profiling service |
| /attack/<id> | GET | | Details of the attack with given <id> |
| /attack/<id> | PATCH | | Update details of the attack with <id> |
| /attack/<id> | DELETE | | Delete the attack with <id> |
| /attack/source/device/<id> | GET | | Retrieve the list with attacks for specific device <id> as source of the attack. |
| /attack/target/device/<id> | GET | | Retrieve the list with attacks for specific device <id> as target of the attack. |
| /compromise | GET | | Retrieve the list with compromised devices. |
| /compromise | POST | | Insert new compromised device to the profiling service |
| /compromise/<id> | GET | | Details of the compromised device with given <id> |
| /compromise/<id> | PATCH | | Update details of the compromised device with <id> |
| /compromise/<id> | DELETE | | Delete the compromised device info with <id> |
| /compromise/device/<id> | GET | | Retrieve the list with compromised devices for specific device <id> |

The following tables present the necessary fields for each entity (end-point) that we described. In particular, Table 4 about the User fields, Table 5 about the Device's fields and Table 6 about the Smarthome.

Table 4 User fields in Profiling Service

| Field | Type | Description |
|----------------------|----------------------|---|
| firstname | String | First name of the user |
| lastname | String | Last name of the user |
| dateofbirth | Date | Date of birth |
| deleted | Boolean | A Boolean field that indicates if a user is deleted (after deletion, data may be kept according the retention policy) |
| email | String | Email of the user |
| gender | String | Enum indicates the gender |
| roles | Array (String) | The role of the user. This will be considered for access rights |
| telephone | String | The telephone of the user. |
| username | String | Username. Unique value |
| devices | Array (JSON Objects) | List of devices that owned by the user |
| aas_reference | Integer | Reference of user to the Cyber-Trust AAS module. |

Table 5 Device fields in Profiling Service

| Field | Type | Description |
|--------------------|--------|--|
| type | String | Type of the device (i.e. smartphone, gateway, etc) |
| description | String | Description of the device |

| | | |
|---------------------------------|--------------|--|
| owner | JSON Object | JSON object about the owner of the device |
| owner.id | String (Ref) | The id of the device owner |
| owner.type | String | The type of the device owner (i.e. individual, organization) |
| user | String (Ref) | The user of device. Sometimes user and owner are different. |
| cpe | Array | List of CPEs related with device |
| device_info | JSON Object | Device info based on monitoring data where the details about the device are stored |
| device_info.hardware_id | String | |
| device_info.model | String | |
| device_info.manufacturer | String | |
| device_info.os | JSON Object | The details of the OS system |
| device_info.os.version | String | |
| device_info.os.name | String | |
| device_info.os.sdk | String | |
| smarthome | String (Ref) | Reference to the smarthome that device belongs. |

Table 6 Smarthome fields in Profiling Service

| Field | Type | Description |
|--------------------|--------------|---|
| name | String | Name of the smart home |
| description | String | Description of the smart home |
| owner | String (Ref) | Reference to the owner user of the smart home |

The required fields for the monitoring data are presented in the next section where the implemented tools are described. However, in order to submit to Profiling Service data, some additional fields are required. The message structure for posting monitoring data is presented in Table 7.

Table 7 Monitoring Data message structure

| Field | Type | Description |
|------------------|--------------|---|
| type | String | The type of data (i.e. performance, netstats, lsof, hash, etc.) |
| device_id | String (Ref) | The ID of the device that sends data |
| data | JSON Object | Data field is a json object containing the monitored data |

Furthermore, vulnerabilities are subject of the integration / synchronization process with the eVDB (A09) and the details are presented in the following paragraphs.

2.2 Integration and synchronization with other data sources

As we already mention the Profiling Service is responsible for two type of data sources: (a) the data that are generated by the devices (or related tools) about the monitoring of devices and, (b) the data that are not directly related with specific devices but with devices in general. The second type of data will be used by the Profiling Service in order to correlate them with the registered devices. Two data sources are considered in this case, the eVDB (A09) database which is a Cyber-Trust module (part of the Profiling Service) and the Patch

DB (separate database that is hosted under the Profiling Service) which is mainly based on available repositories for software updates, patches and firmware updates.

2.2.1 Synchronization with eVDB (A09)

The eVDB (A09) Integration and Synchronization Service is built around the basic MISP entity types (MISP Events, Attributes, Objects & Galaxies) and is broken down in two synchronization types; Initial & Regular.

- **Initial Synchronization:** This type of synchronization should only take place once, right after the deployment of eVDB ISS service. Completion time mainly depends on the amount of data on the eVDB (A09) current instance. Furthermore, the initial synchronization can be parametrized to fetch and sync past events.
- **Regular/Periodic Synchronization:** This type of synchronization is accomplished by scheduling repetitive system tasks (CRONS) with crontab files. With this process, the eVDB is synced almost real time (based on the frequency of execution). The regular service can be also updated to utilize any message queues available for real-time sync.

The eVDB ISS reads all records for each eVDB (A09) entry based on MISP (Event, Attribute etc) and creates them in the Profiling Service. Note, this module of profiling service is independent from any device profile as is common for all the purposes of profiling service.

The integration of Profiling Service and eVDB is done through the issuing of an authorization key through by the eVDB (A09). This key is then provided as Authorization Header to all requests to eVDB (A09) API. The communication approach between the two components is direct without the intervention of the Integration BUS of Cyber-Trust.

During regular synchronization flow, the main component searches for the past day's new or updated data (MISP Events, Attributes, Objects etc). Note that record purification is the process of detaching duplicate and/or irrelevant information from the eVDB API's responses before persisting the data on the Profiling Service. For example, purifying an Event record object returned from the eVDB (A09) API before creating it on the Profiling Service, would remove the nested MISP Objects and their attributes from that object.

The eVDB ISS is hosted on the profiling service resources. Currently, we are syncing all the available data without any filtering and selection of fields of interest. This may change after the first pilot phase of Cyber-Trust in order to keep only the necessary for the device's fields.

As we mentioned at the introduction of eVDB ISS, four new entities are prepared in the profiling service:

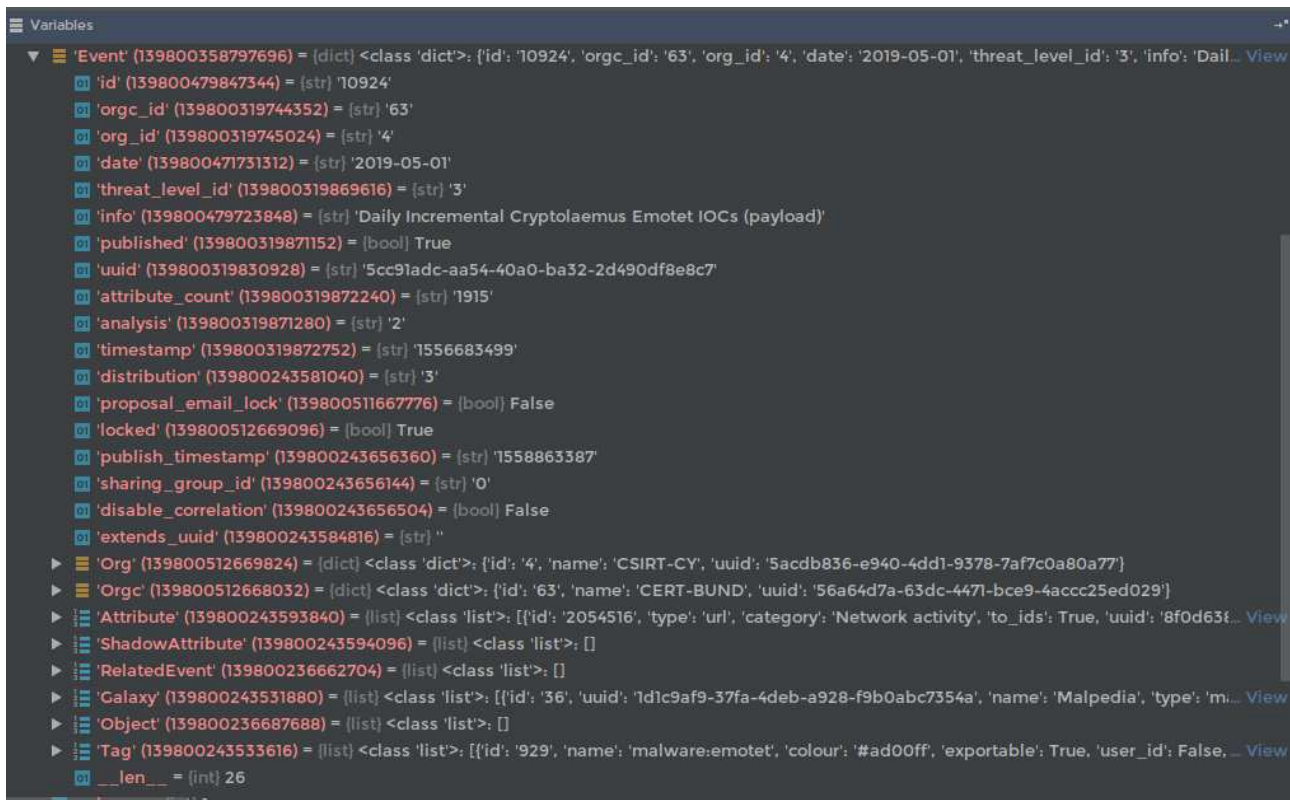
- **Events:** Vulnerability/event
- **Attributes:** Vulnerability/Attribute
- **Objects:** Vulnerability/Object
- **Galaxies:** Vulnerability/Galaxy

Table 8 Profiling Service end-points for eVDB (A09)

| End-point | Method | Description |
|-------------------------------|--------|--|
| /vulnerability/event | GET | Retrieve the list with eVDB synced events. |
| /vulnerability/event | POST | Insert new eVDB event to the profiling service |
| /vulnerability/event/<id> | GET | Details of the eVDB event with given <id> |
| /vulnerability/event/<id> | PATCH | Update details of the eVDB with <id> |
| /vulnerability/event/<id> | DELETE | Delete the eVDB event with <id> from the Profiling Service |
| /vulnerability/attribute | GET | Retrieve the list with available eVDB attributes. |
| /vulnerability/attribute | POST | Insert new eVDB attribute to the profiling service |
| /vulnerability/attribute | GET | Details of the eVDB attribute with given <id> |
| /vulnerability/attribute/<id> | PATCH | Update details of the eVDB attributes with <id> |

| | | |
|-------------------------------|--------|--|
| /vulnerability/attribute/<id> | DELETE | Delete the eVDB attribute with <id> from the Profiling Service |
| /vulnerability/object | GET | Retrieve the list with available eVDB objects. |
| /vulnerability/object | POST | Insert new eVDB object to the profiling service |
| /vulnerability/object/<id> | GET | Details of the eVDB object with given <id> |
| /vulnerability/object/<id> | PATCH | Update details of the eVDB object with <id> |
| /vulnerability/object/<id> | DELETE | Delete the eVDB object with <id> from the Profiling Service |
| /vulnerability/galaxy | GET | Retrieve the list with available eVDB galaxies. |
| /vulnerability/galaxy | POST | Insert new eVDB galaxy to the profiling service |
| /vulnerability/galaxy/<id> | GET | Details of the eVDB galaxy with given <id> |
| /vulnerability/galaxy/<id> | PATCH | Update details of the eVDB galaxy with <id> |
| /vulnerability/galaxy/<id> | DELETE | Delete the eVDB galaxy with <id> from the Profiling Service |

While the sync is based on complex Events JSON objects (including all the relevant to event information), a parser is used in order to extract the attributes, objects and galaxies that may include in the Event Object and stored them in separate data entities. As a result of this process, is the better data management within the profiling service where the search and correlation with registered devices can be done in a more efficient way. Table 8 presents a set of end-points that are used from the eVDB ISS to insert data to the profiling service.



```

Variables
Event (139800358797696) = {dict} <class 'dict': {'id': '10924', 'orgc_id': '63', 'org_id': '4', 'date': '2019-05-01', 'threat_level_id': '3', 'info': 'Daily Incremental Cryptolaemus Emotet IOCs (payload)', 'published': True, 'uuid': '5cc91adc-aa54-40a0-ba32-2d490df8e8c7', 'attribute_count': '1915', 'analysis': '2', 'timestamp': '1556683499', 'distribution': '3', 'proposal_email_lock': False, 'locked': True, 'publish_timestamp': '1558863387', 'sharing_group_id': '0', 'disable_correlation': False, 'extends_uuid': '', 'Org': {'id': '4', 'name': 'CSIRT-CY', 'uuid': '5acdb836-e940-4dd1-9378-7af7c0a80a77'}, 'Attribute': [{'id': '2054516', 'type': 'url', 'category': 'Network activity', 'to_ids': True, 'uuid': '8f0d63f...'}, 'ShadowAttribute': [], 'RelatedEvent': [], 'Galaxy': [{'id': '36', 'name': 'Malpedia', 'type': 'malware.emotet', 'colour': '#ad00ff', 'exportable': True, 'user_id': False, ...}], 'Object': [], 'Tag': [{'id': '929', 'name': 'malware.emotet', 'colour': '#ad00ff', 'exportable': True, 'user_id': False, ...}], '__len__': 26}

```

Figure 3 Example of event processing during eVDB (A09) sync

2.2.2 Patch DB

The Patch DB of Cyber-Trust project is hosted on Profiling Service (A17) and is accessible through a dedicated REST API (Part of Profiling Service REST API). Taking into advantage the Profiling Service back-end technology infrastructure and the internal security modules, the isolation of data between different data classes is

applicable. As a result of this, the Patch DB data are isolated with the Devices as well as with the Evidence DB (subject later in the document).

The Patch DB is consisted from two main storage systems, the binary storage and the metadata storage. Each binary (patch) has the relevant metadata record including information about the patch/update as well as the binary MD5 signature.

2.2.2.1 Patch DB Crawling Service

The data acquisition for the Patch DB is based on the Patch DB Crawling Service that has been developed and deployed on Patch DB module under the Profiling Service. In particular, a python-based service has been prepared for periodic search and collect information from specific public sources. With a list of public websites or data repositories as an input, the service is trying to detect binaries (software updates) and the targeting device for storing to the Patch DB.

The following list presents examples of firmware sources that have been integrated with the Cyber-Trust Patch DB.

- SAMSUNG: <https://www.sammobile.com/firmwares/>
- LG: <https://www.lg.com/uk/support/software-firmware>
- XPERIA: <https://xperiafirmware.com/>
- Other Sources: <https://www.needrom.com/>

| Country/Carrier | Date | Version | PDA | CSC |
|--------------------|------------|---------|---------------|---------------|
| United Kingdom | 2019-12-23 | 8.0.0 | G935FXXS7ESL5 | G935FOXA7ESL5 |
| France (Orange) | 2019-12-19 | 8.0.0 | G935FXXS7ESL3 | G935FFTM7ESL3 |
| Poland (Orange) | 2019-12-19 | 8.0.0 | G935FXXS7ESL3 | G935FOPV7ESL3 |
| Spain (Orange) | 2019-12-19 | 8.0.0 | G935FXXS7ESL3 | G935FAMO7ESL3 |
| Romania (Orange) | 2019-12-19 | 8.0.0 | G935FXXS7ESL3 | G935FORO7ESL3 |
| Croatia (T-Mobile) | 2019-12-10 | 8.0.0 | G935FXXS7ESK1 | G935FDHX7ESK1 |
| Croatia (Bonbon) | 2019-12-10 | 8.0.0 | G935FXXS7ESK1 | G935FDHX7ESK1 |
| Poland (Heyah) | 2019-12-10 | 8.0.0 | G935FXXS7ESK1 | G935FDPX7ESK1 |
| Poland (T-mobile) | 2019-12-10 | 8.0.0 | G935FXXS7ESK1 | G935FDPX7ESK1 |

Figure 4 Example of Samsung Firmware Repository

Figure 4 presents an example of the Samsung Firmware repository¹. Based on the available information, a link or the binary of the update itself is download and synced with the Patch DB.

Patch DB is available under the Profiling Service through the following URL:

Patch DB: <https://profiling-service/patchdb>

The end-point is a REST API with the following POST and GET Methods available. In particular, there are three available end-points that are presented in Table 9. Note that update of records is not allowed, thus no available end-point for update/delete data are available.

¹ <https://www.sammobile.com/firmwares/>

Table 9 Patch DB REST API end-points

| End-point | Method | Description |
|----------------------|--------|---|
| /patchdb | POST | Insert new patch/update in the Patch DB |
| /patchdb/<id> | GET | Get patch details based on the record with <id> |
| /patchdb/binary/<id> | GET | Retrieve the binary details of the patch <id> |
| Download binary | GET | Temporary link that can be found in binary details for downloading the patch. |

Table 10 Patch DB fields

| Field | Type | Description |
|-------------------|-------------------------|--|
| model | String | The model of the device |
| model_name | String | Description of the model name |
| vendor | String | The vendor of the device |
| version | String | Version numbering of the patch |
| date | Date (ISO 8601) | Release date of the patch |
| os | String | OS that the patch is applicable |
| cpe | String | Calculated CPE name for the patch |
| reference | String | Reference to the source of the patch |
| url | String | URL for direct download from the manufacturer |
| binary | Binary Reference (uuid) | Reference of the available binary. The location is the binary part of the Patch DB |
| info | JSON Object | JSON Object that contains additional information about the patch record. The information includes all the additional available data from the manufacturer/vendor |

Data can be insert in the Patch DB using the available end-point (POST Method) with body a json message based on the fields that are presented in Table 10. Similar with profiling service, the call should be authorized using the Cyber-Trust Authorization and Authentication Service and only specific user roles and components can access the data.

2.2.3 Data Correlation

The Data Correlation Engine is a service provided by the Profiling Service and is responsible to correlate data between the devices and the external sources (eVDB and Patch DB). In case of new data, either from the eVDB or Patch DB integration and crawling services, the Data Correlation Engine will be notified in order to identify the affected devices. In particular, the purpose of the correlation engine is to match the available and registered devices of Cyber-Trust platform with the list of vulnerabilities.

The correlation is mainly based on the search engine of the Profiling Service. The search is applicable for all the entities of the Profiling Service including eVDB Sync, Patch DB, CPE and device related data. For the correlation, we are searching the fields such as CPE strings, device types and software information. The result of the correlation engine can be stored in two ways: (a) the enhancement of the Device Object (i.e. in case of new vulnerability, the device object will be enhanced to include the reference of vulnerability and (b) the correlation matrix, a JSON object that includes the correlated fields with the relevant metadata. In both cases, the outcome of the correlation is published to the relevant Cyber-Trust components as well as the users of interest through the Integration BUS. Currently, two correlation cases are available under the profiling

service. The correlator engine is triggered when data from eVDB ISS and Patch DB Crawling Service are available in Profiling Service.

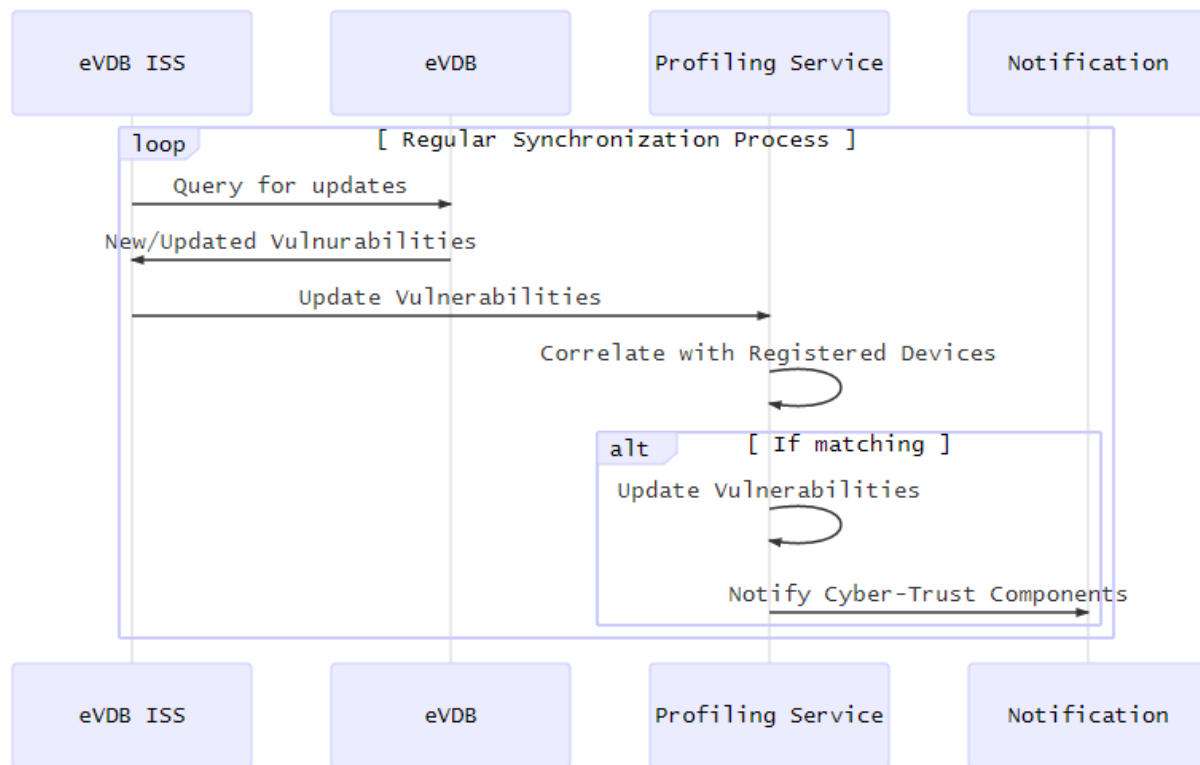


Figure 5 eVDB ISS Workflow in Profiling Service

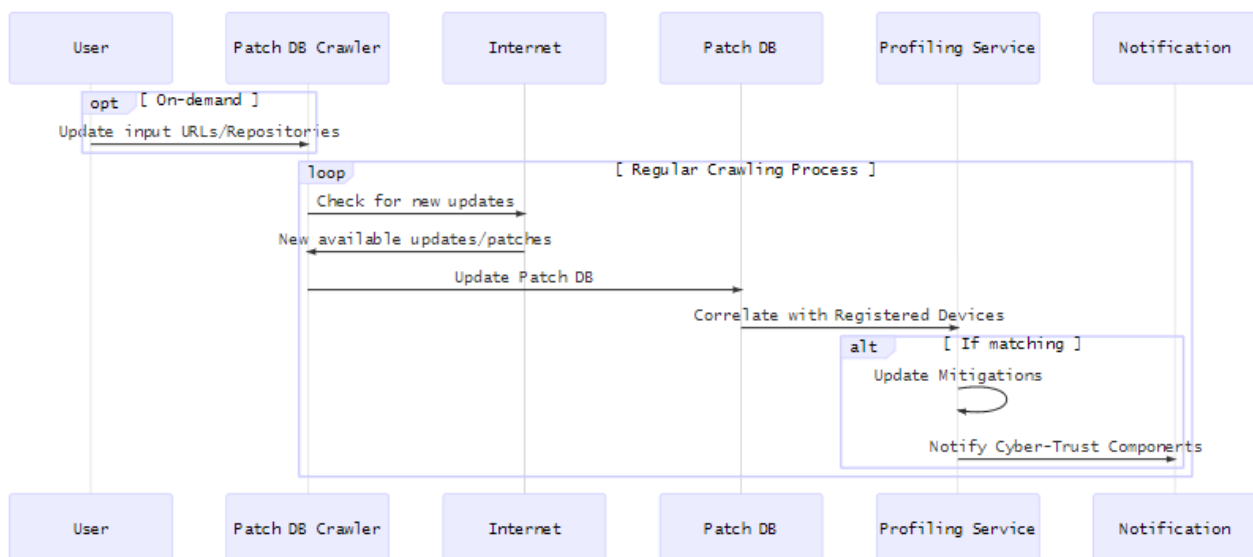


Figure 6 Patch DB Crawler - Correlator Workflow

Figure 5 and Figure 6 present two workflows, about eVDB and Patch DB services, where the Profiling Service is responsible to correlate the new information related with vulnerabilities and updates with the list of registered devices.

For the case of vulnerabilities, the Vulnerability entity is responsible to keep the result of the correlator. In particular, the fields presented in Table 11 describe the result of the correlation between new vulnerabilities and registered devices.

Table 11 Vulnerability fields in Profiling Service

| Field | Type | Description |
|-----------------------------|--------------|---|
| id | String | The unique ID of the vulnerability in Profiling Service |
| evdb_ref | String | The reference of vulnerability source in eVDB system |
| cve | String | The CVE number of the vulnerability |
| device_id | String (Ref) | Reference to the device id if it is affected by the vulnerability |
| mitigation_type | String | The type of mitigation that has been applied to the specific vulnerability (This field will be a result of an update of the record) |
| mitigation_reference | String (Ref) | Reference to the mitigation record within the Profiling Service |

The mitigation fields of Table 11 will be added in the record of vulnerability after Cyber-Trust components investigate, analyses and propose mitigation actions. The management of vulnerabilities in Profiling Service is established through the REST API end-points presented in Table 12.

Table 12 Vulnerabilities REST API end-points

| End-point | Method | Description |
|----------------------------|--------|---|
| /vulnerability | GET | Retrieve the list with vulnerabilities. |
| /vulnerability | POST | Insert new vulnerability to the profiling service |
| /vulnerability/<id> | GET | Details of the vulnerability with given <id> |
| /vulnerability/<id> | PATCH | Update details of the vulnerability with <id> |
| /vulnerability/<id> | DELETE | Delete the vulnerability with <id> |
| /vulnerability/device/<id> | GET | Retrieve a list with vulnerabilities related with device ID |

2.3 Vulnerabilities, Compromised Devices, Attacks and Mitigations

In order to record Cyber-Trust results, three entities have been created under the Profiling Service, the correlated vulnerabilities, the compromised devices, the detected attacks and the mitigations with reference to device and vulnerabilities.

Table 13 presents the required fields to describe a compromised device. The record for compromised device is a result of processing of generated alerts as well as outcome from other Cyber-Components. Table 14 presents the fields that are related with an attack targeting or launched by a registered device. Finally, Table 15 presents mitigation details and the correlation with a recorded vulnerability in the Profiling Service.

Table 13 Compromised Devices fields in Profiling Service

| Field | Type | Description |
|------------------|---------------|--|
| id | String | The unique ID of compromise in Profiling Service |
| status | String | Status of compromised device. Active, Completed, etc. |
| type | String (ENUM) | The type of the event that set the device as compromised. Such types are criticalFileAltered, firmwareHack, OS_Hack, backdoorService, maliciousCode, dataExfiltration dataDestruction. |
| device_id | String (Ref) | Reference of the compromised device, device id. |

| | | |
|------------------------|--------------|---|
| detector | String | Details about the detection of the event that set the device as compromised. |
| alerts | Array [Refs] | Reference to alerts that are linked with compromised devices |
| additional_info | String | Additional information related with the compromised device and the compromise type. |

Table 14 Attack fields in Profiling Service

| Field | Type | Description |
|------------------------|---------------------|--|
| id | String | The unique ID of attack in Profiling Service |
| status | String | Status of the attack. Active, Completed, etc. |
| type | String (ENUM) | The type of the event that set the device as compromised. Such types are remoteExploit, DoS, DDoS, enumeration, etc. |
| target_devices | Arrey [<device_id>] | Reference of the devices that are targeted by the attack. |
| source_device | String [Ref] | Reference to the source of the attack device. None if not available. |
| start_timestamp | Date (ISO 8601) | The time that the attack was detected to begin |
| end_timestamp | Date (ISO 8601) | When the attack ceased |
| detector | String | Details about the detector of the attack that set the device as compromised. |
| additional_info | String | Additional information related with the attack. |

Table 15 Mitigation fields in Profiling Service

| Field | Type | Description |
|-------------------------|---------------|---|
| id | String | The unique ID of the mitigation in Profiling Service |
| status | String | Status of mitigation. Ready, Applied, etc. |
| type | String (ENUM) | The type of the mitigation. Such types are fullPatch, ISR, antecedentInvalidation, etc. |
| vulnerability_id | String (Ref) | Reference to the vulnerability that the mitigation is related. |
| additional_info | String | Additional information related with the mitigation. |

2.4 Monitoring Rules, Alerts and Notifications

The most important part of the Profiling Service is the capability to apply rules and correlate data. This process will result in events related with the activity of devices and through notification messages will be broadcasted or directly sent to the Cyber-Trust components and the end-user (i.e. ISP Operator, Admin, Device Owner, etc.).

2.4.1 Rules and Alerts

The Profiling Service is responsible for the storage of rules and alerts about the devices. Within the rules model, the stored information is related with the definition of the rule including the conditions while the alerts model is responsible to keep the generated message when a rule is fired by the device or the comparison component. In both cases, the alerting data are kept in the Profiling Service.

The basic fields that need to be provided in order to create a new rule are presented in Table 16.

Table 16 Rule fields in Profiling Service

| Field | Type | Description |
|-----------------------------------|---------------------|---|
| id | UUID Object | The unique ID of the rule in the profiling service |
| name | String | The name of the rule |
| description | Text | The description of the rule. Free text that will provide more details. |
| device_type | Array of Strings | Each string within the array represents a class of devices that the rule is applicable. |
| device_id | UUID Object or Null | In case that the rule is for specific device the Device ID will be provided. Otherwise the field will be null |
| condition | | Array of conditions of the rule |
| condition.terms | | Array with the clauses of the condition |
| condition.terms.key | | Key of the clause |
| condition.terms.value | | Value of the rule |
| condition.terms.operator | | Operator |
| condition.logical_operator | | |

To create a new rule in the system, a JSON file with the fields above should be prepared and posted to the Profiling Service through the relevant Rules end-point.

The above rules are consumed from the devices in the case that the monitoring service is responsible for the application of rules and generation of alerts. Otherwise, the dedicated service for the application of rules on the monitoring data is synced with the Profiling Service in order to fetch any possible updates on the rules. The result of the monitoring process including the filtering of data based on rules is an ALERT that will be generated and published to the Cyber-Trust Platform. Alerts are firstly posted on the Profiling Service with all the relevant data and then, the Profiling Service is responsible to notify the Information BUS and the relevant components.

Similar with the rules, alert data are stored in the profiling service under the Alerts entity. The necessary fields for an alert are provided in the Table 17.

Table 17 Alert fields in Profiling Service

| Field | Type | Description |
|--------------------|---------------|---|
| id | String | The unique ID of the alert |
| alert_type | String (ENUM) | Type of the alert. This is related with the rule that triggered the alert |
| device_id | String (Ref) | The device that cause the raise of the alert |
| device_type | String | The type of the device. i.e. smartphone, etc. |

| | | |
|-----------------------|--------------|--|
| reason | JSON Object | The reason that causes the alert is a JSON object with rule and other details. |
| reason.rule_id | String (Ref) | The ID of the rule that triggers the alert |
| reason.message | String | An automated message based on the rule and the values of metrics |
| reason.code | Integer | A standard coding (for Cyber-Trust project) of the reason that causes the alert. |
| criticality | String | Criticality metric that characterizes the alert |
| importance | String | Importance metric that characterizes the alert |

2.4.2 Notification messages

Within the Cyber-Trust platform, Profiling Service is responsible to provide data related with devices (monitoring, alerts) to the rest of Cyber-Trust platform including the end-user. The majority of the cases, notifications are linked with the alerts that are generated either by devices or other related components.

There are two types of notification messages:

- Component to Component Notifications
- Component to End-User Notifications

2.4.2.1 Component to Component Notifications

The Profiling Service through the available communication approaches (Pub/Sub or Direct messages) is responsible to produce notification messages related with device activities in order to inform other Cyber-Trust components. Such cases are the notification of Trust Management System, Cyber Defense Service, etc. The main notification channel of this kind of communication is the Integration BUS. Using the appropriate topics, the notification will reach the components of destination. However, direct messages to the components of destination can be sent.

An example is the communication between Profiling Service and the Trust Management Service. The type of messages is listed below. Detailed description of messages including all the relevant components will be presented in the deliverable D8.2 “Integrated CYBERTRUST platform & testing: v1”.

- **New device installed:** Implementation is done under the create new device end-point
- **Device removed:** Implementation is done under the update or delete device end-points
- **New vulnerabilities associated with device(s):** The result of the correlation between eVDB service and devices, vulnerability entry as presented in Table 11, is communicated with the TMS.
- **Device vulnerabilities mitigated:** When a mitigation is applied to a device and the vulnerability entry is updated accordingly, a notification is sent to the TMS.
- **Device compromised:** When an alert arrived at the system and the possible attacked is described, the device compromised message is prepared and sent to the TMS.
- **Device health restored:** A device that has been previously compromised is detected to have its health restored and the status is updated in the Profiling Service. TMS will be informed for this action through a message.
- **Device deviation from nominal metrics:** Alert activity as a result from the application of rules will be sent to the TMS as device deviation from nominal metrics message.

- **Device return to nominal metrics:** When the status of device (considering the monitoring metrics) returned to normal values, the TMS will be notified.
- **Device detected to launch attack:** When a device has been detected to launch attacks against other devices, the Profiling Service records the attack and the TMS will be informed.
- **Device detected to be the target of attack:** When a device has been detected to be the target of attacks from other devices, the Profiling Service records the attack and the TMS will be informed.

2.4.2.2 Component to End-User Notifications

The second type of notification messages has as its final destination the end-users. The end-users will be notified about the activity of their devices with respect to Cyber-Trust technological tools and processes through the Cyber-Trust UI, the Cyber-Trust Mobile APP as well as other communication means such as SMS. More details about the notification channels will be delivered in D8.2 “Integrated CYBERTRUST platform & testing: v1”.

For the implementation of Component to End-User notifications, we are using three communication means: (a) the Integration BUS, (b) WebSocket services and (c) Google’s Firebase Notification Service.

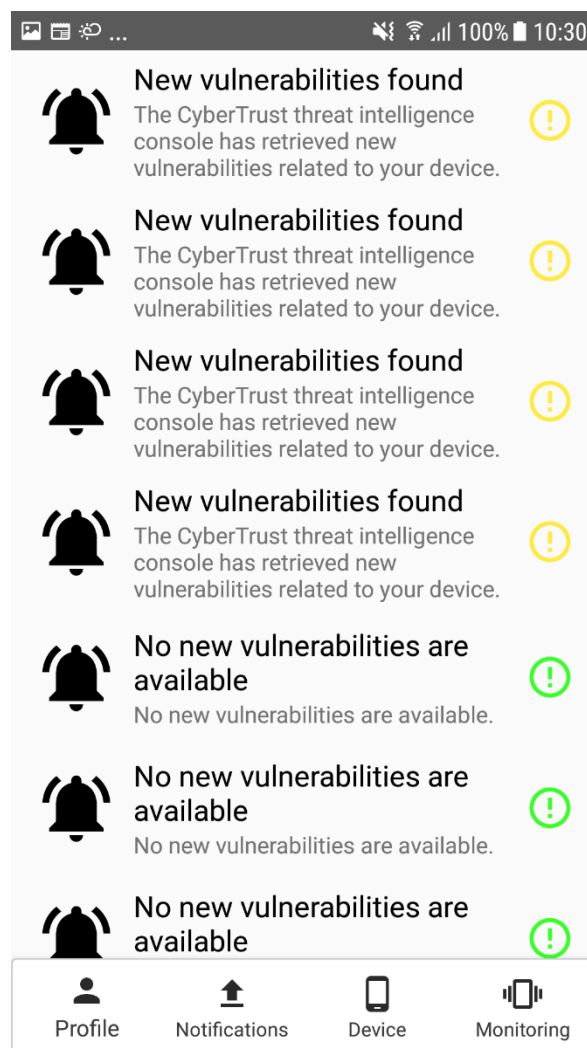


Figure 7 Notifications page on Cyber-Trust Mobile APP

3. IoT Device Monitoring

Due to its intended operation, the SDM (A03m) is designed to check whether the hosting device performs as intended by its manufacturer, ensures that critical OS files are uncompromised and that only secure means of communication are used. Monitoring services have been developed and executed on devices for the acquisition of monitoring data. Data regularly synched with the Profiling Service involve information regarding runtime processes and used hardware resources. Only in the case of identified suspicious traffic and activity, network packages are signed by SDM (A03m) and communicated with the CT Cyber Defense service for further investigation. The detection of suspicious traffic and activity at device level is based on the rules that have been configured and stored in the Profiling Services.

Considering the hosting OS, three different SMB (A03m) implementations will be developed within Cyber-Trust to accommodate the following classes of IoT devices:

1. Android-Based OS Devices (Smartphones and Smart TVs)
2. Devices running a Linux-/Windows-based OS distribution
3. Devices implemented to use IoT cloud Services

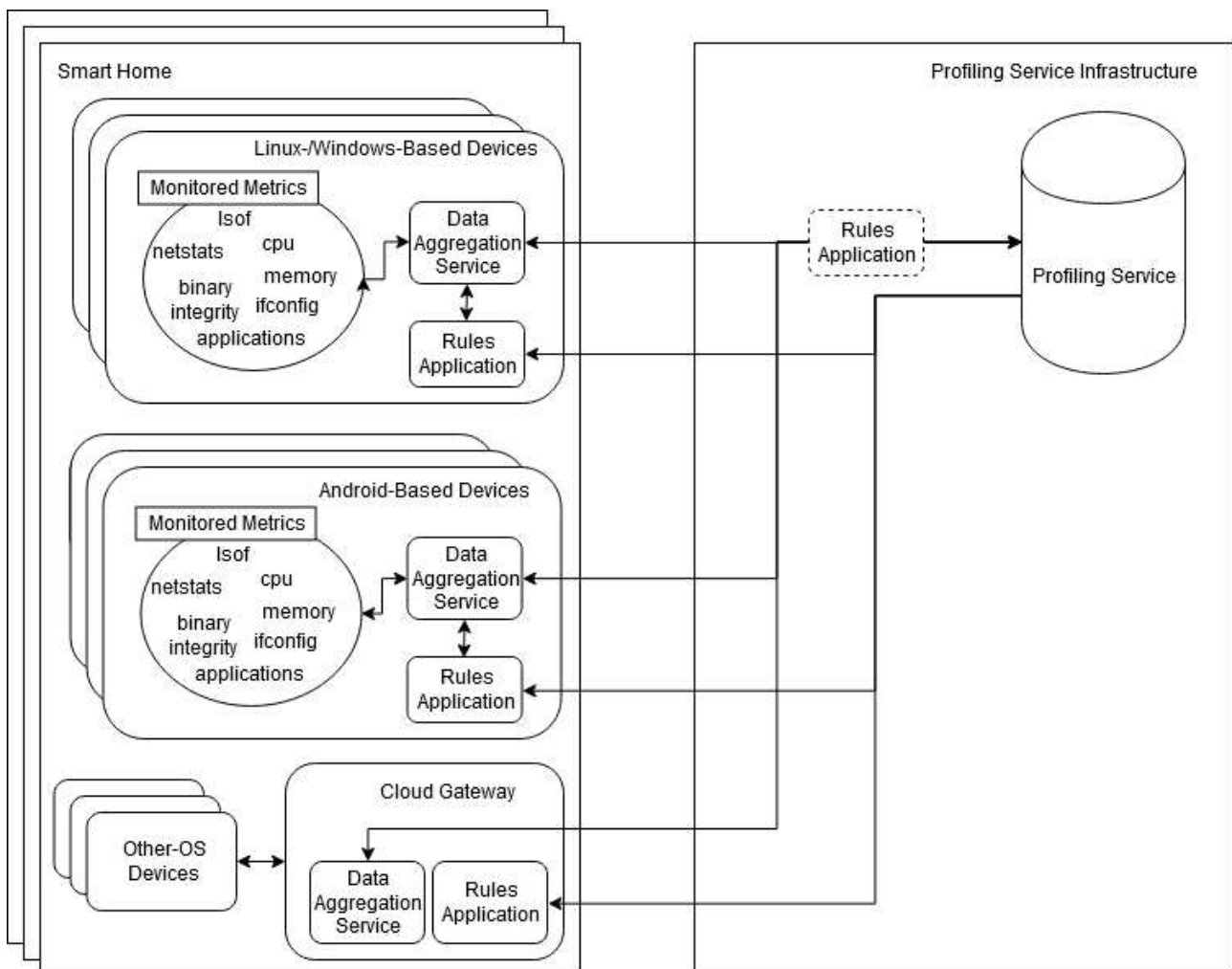


Figure 8 High Level diagram of Monitor Activity

Regardless the type of class of device, the detection of abnormal behavior is based the rules that are configured and stored in the Profiling Services and they are associated with the type of devices and user preferences. To implement this, the comparator component, part of the SDM (A03m), is developed in order to filter the data by applying the rules on the captured data in real-time. In the Cyber-Trust project device

tools, two approaches for monitoring and filtering the data are implemented: (a) the comparator is hosted and run on the device (Figure 8 the Rules Application component at device layer), and (b) the comparator is hosted centrally and serving multiple devices (Figure 8 the dotted Rules Application component at Profiling Service infrastructure). The first approach imposes overheads at device levels with the detection of abnormal activity is performed with the minimum latency while the second approach imposes minimum computational overheads with the drawback of the increased latency. However, as the scope of monitoring is the alert and notification of user and the other Cyber-Trust components, the latency can be considered similar. While the centralized solution seems better, the availability of the device level solution is mandatory as maximize the autonomy of the components and minimize the dependencies with off-device tools. At this stage of the project, the choice of the approach is done manually.

For each class of devices, the monitoring metrics/data have been classified in three categories: (a) the metrics related with performance such as CPU and Memory usage, (b) the metrics related with network such as open ports and, (c) the monitoring of binary files to ensure the integrity of the firmware and of the critical OS files.

3.1 Android-Based OS Devices (Smartphones, Smart TVs)

To enable the monitoring of smart phones that are running a Mobile operating system such as Android OS (or iOS) as well as Smart TV OS (e.g. Android TV) we proceeded to the design and implementation of a Cyber-Trust Mobile App. The smart device monitoring app is implemented only for Android OS using native Android instead of a hybrid framework to cover as many IoT devices as possible. The reason of following this approach is that the considered hybrid frameworks (such as Ionic Framework) do not allow access to operating system controls and therefore the functionality of monitoring cannot be supported with the use of these frameworks. Furthermore, the usage of Cyber-Trust mobile app is two-fold. Firstly, we implemented all the monitoring functionalities that are required under the Cyber-Trust project and secondly, we developed a user-based UI (Mobile APP) that enable the monitoring of Cyber-Trust tools as well as the receiving of the alert notification that are generated by the Cyber-Trust system. Similar with the Linux-based devices and monitoring services, the filtering of monitoring data (apply rules) can be done on-device or off-device.

The three categories of monitoring data that mentioned and described for the case of Linux-based are also monitored in the case of smart devices. Details of the fields and the tools that deployed for acquiring the monitoring data are presented in the following paragraphs.

3.1.1 Performance Monitoring

Performance issue are highly related with anomalies that may appeared on the devices. There are cases where the root of cause of such issues are related with the user usage, the performance of applications themselves or capacity and hardware limitations. However, performance issues are symptoms of malware running on the device (abnormal behavior) that consumes hardware resources. As a result of this, the necessity for performance monitoring of device towards the detection of malwares and other cyber-security related threats is an important process.

Table 18 presents a set of performance monitoring fields that are captured from the mobile app application.

Table 18 Performance Monitoring fields for Devices

| Field | Units | Description |
|----------------------|----------------|--------------------------------------|
| cpu_overall | Percentage (%) | Overall CPU Usage |
| cpu_user | Percentage (%) | CPU utilization by user applications |
| cpu_system | Percentage (%) | CPU utilization by system services |
| ram_usage | Percentage (%) | Percentage of RAM usage |
| ram_total | Bytes | Size of RAM in bytes |
| ram_available | Bytes | The available RAM in bytes |

| | | |
|-------------------------------|----------------|--|
| internal_storage_usage | Percentage (%) | Usage of internal storage (1- Available/Total) |
| external_storage_usage | Percentage (%) | Usage of external storage (1- Available/Total) |

The acquired performance monitoring fields (metrics) are periodically collected from the smart device using the JAVA library/class for Android Services.Usages. The frequency of retrieve the information is configurable with default period at 5 seconds. The developed mobile service is able to calculate the average value of each metrics the last 1, 5 or 15 minutes.

Through a REST API call, the performance monitoring service requests the applicable for the device rules that are applied in real-time on the monitored data. When a rule is fired, the appropriate alert message is constructed and published to the Profiling Service and then to the Cyber-Trust platform.

For every monitoring cycle, a JSON message including the above fields, the current time and the device ID is prepared. The JSON Object is posted to the profiling service in two cases. In the first case the application of rules is done on the device (embedded) and data are sent to the profiling service when a rule is fired for storage and usage from the evidence DB and other Cyber-Trust components. Alternative, data are sent in real-time to a centralized component and profiling service to apply the rules.

The mobile app has a dedicated page for visualization of the current values of performance metrics Figure 9.

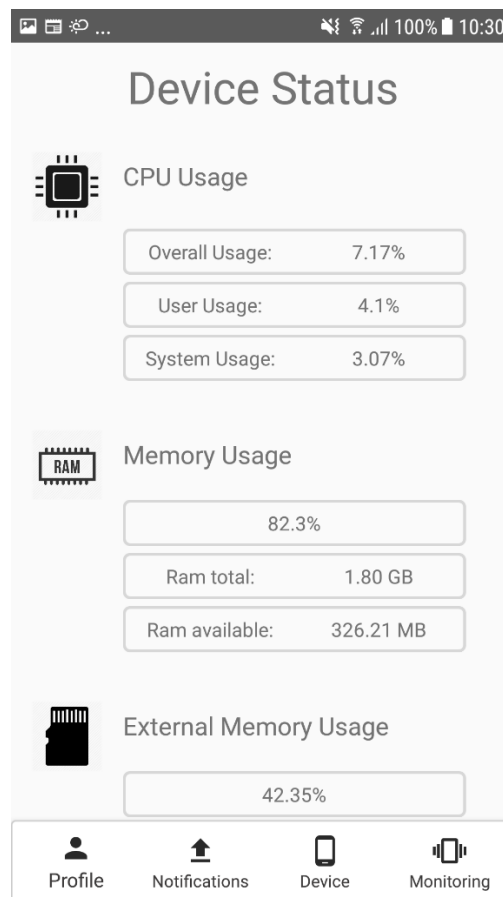


Figure 9 Device Status Page (Performance Monitoring)

3.1.2 Critical OS and Firmware Integrity Monitoring

An important parameter that is required for the normal operation of a device is to check the whether the hosting device performs as intended by its manufacturer by ensuring that the critical OS files are uncompromised and the firmware integrity. In Cyber-Trust project, we developed methodologies and scripts that are mainly based on the calculation of the hash of critical files in the entire device system followed by a

secure storage of calculated hashes. Calculating the hashes in regular intervals allow the monitoring and detection of possible changes as well as replacements of such files under a malicious and suspicious activity. To enable the Critical OS and Firmware Integrity monitoring on the mobile app, a function is developed which receives as input a list of critical files/binaries and for each one the calculation of a hash is performed. The calculated hashes are communicated with Profiling Service where the comparing with previous hashes is taking place. In case of mismatch, the appropriate alert is generated and the relevant Cyber-Trust components are informed.

The list below indicates some examples of monitoring Critical OS files. The list can be amendment including additional files and other binaries that should be monitored.

- /proc/cpuinfo
- /proc/version
- /proc/meminfo
- /proc/ioports

The selection of the files is related with the monitoring activity in order to ensure that the binaries/executables responsible for data extraction are not altered in the process of an attack.

Table 19 Critical OS Files Integrity Check fields for Devices

| Field | Type | Description |
|------------------------|--------|--|
| filename | String | The name of the file that has been checked |
| path | String | The exact location of the file (path) |
| hash | String | The calculated has |
| algorithm | String | The algorithm that is used for calculating the hash (i.e. MD5) |
| additional_info | String | Additional information that may exists and should be recorded |

The implementation of the MD5 calculation is based on “*java.security.MessageDigest*” class and “*getDistance(“MD5”)*” function. Table 19 presents the required fields for the monitoring of critical OS files integrity. The monitoring of critical files is to ensure that important binaries of the operating and more system are not compromised. However, the developed mobile application monitors the system information and the status of installed application.

Table 20 presents as set of fields that are currently monitored by the application.

Table 20 Status and About fields for Devices

| Field | Type | Description |
|-----------------------------|--------|---|
| device_id | String | The device ID by manufacture |
| device_model | String | The device model |
| device_manufacturer | String | The device manufacturer |
| os_version | String | The OS version |
| os_build | String | The OS build version |
| os_api | String | The version of Android OS API |
| os_build_date | String | The unique device ID as provided by manufacturer. |
| device_serial_number | String | The device serial number |
| device_imei | Array | List of available devices’ IMEI |

For the installed applications, the mobile app is able to extract information about the name of the app and the version. In particular, using the PackageManager class available in Java for Android development, information about the installed Applications and Packages is available for monitoring. The information presented in Table 21 is captured and posted to the profiling service.

Table 21 Installed Application fields for Devices

| Field | Type | Description |
|-------------------------|--------|---|
| app_name | String | The name of the application |
| app_version | String | The version of the current installation |
| app_package_name | String | The package name |
| app_size | String | The current size of the application |

The purpose of monitoring such applications is to be able to search and correlate updates/vulnerabilities that are related with the device. Using the initial information, we construct the CPE and proceed to a matching with the official directory. Then, each profile of the devices includes information about the apps and in case of matching alerts are generated.

While the installed applications are monitored mainly for new updates, the open processes are also part of the monitoring activity. In particular, the usage of **lsot** “list open files” command is done under the monitoring agents. Similar with other tools, the output of the command is processed for the creation of JSON objects with the fields presented in Table 22.

Table 22 LSOF command fields for Devices

| Field | Type | Description |
|----------------|---------|--|
| command | String | The name of the command that is running |
| pid | Integer | The identification number of the process |
| user | String | The package name |
| size | Integer | The current size of the running program |
| node | Integer | The opened file node number |
| path | String | The path of the executable binary |

3.1.3 Device Level Network Monitoring

The most important module(s) of a device with respect cyber-security issues is the ones that are related with the network. As network interfaces are the main interface of data exchange between of a device and the rest of network including local network and internet the monitoring and integration with intrusion prevention and detection systems is mandatory. In Cyber-Trust project, the monitoring of traffic is done under the Smart Gateway Module (SGM) which is described in deliverable D6.3. Monitoring of network activity at device level is mainly focused on open ports and network statistics that can be extracted from the device. Two well-known tools have been utilized for the extraction of network metrics, the netstat and the ifconfig. The execution of the tools is triggered by the SDM and the output result is processed for the extraction of the metrics of interest.

Table 23 presents the fields that extracted from the execution of netstat command with -ntp parameters. The extraction of fields is done using a parser that is developed resulting in an array JSON Object that contains all the active internet connections of the smart device. The JSON is posted on the profiling service for storing under the monitoring data of each device.

Table 23 Mobile App netstat fields

| Field | Type | Description |
|------------------------|---------|--|
| protocol | String | The name of the network protocol (TCP or UDP). |
| recv_q | String | RECV-Q metric tell us how much data is in the queue for that connection (socket), waiting to be read |
| send_q | String | RECV-Q metric tell us how much data is in the queue for that connection (socket), waiting to be sent |
| local_address | String | The IP address of the local computer. An asterisk (*) is shown for the host if the server is listening on all interfaces. |
| local_port | Integer | The port number of the local end (the hosted device). If the port is not yet established, the port number is shown as an asterisk. |
| foreign_address | String | The IP address of the foreign end (remote device). An asterisk (*) is shown for the host if the server is listening on all interfaces. |
| foreign_port | Integer | The port number of the remote computer. An asterisk (*) is shown for the host if the server is listening on all interfaces. |
| state | String | Indicates the state of a TCP connection. The possible states are as follows: CLOSE_WAIT, CLOSED, ESTABLISHED, FIN_WAIT_1, FIN_WAIT_2, LAST_ACK, LISTEN, SYN_RECEIVED, SYN_SEND, and TIME_WAIT. |
| pid | Integer | Process ID of the application/service that established a connection |
| program | String | The name of the program running in the process with that pid. |

The second command that is used under this monitoring class is the ifconfig. Similar with netstat, ifconfig is triggered by the mobile app and is responsible to extract information related to the network interface. Table 24 presents that extracted fields upon the execution of ifconfig command.

Table 24 Mobile App ifconfig fields

| Field | Type | Description |
|--------------------|---------|---|
| rx_packets | Integer | The rx_* fields show the total number of packets received. Additional information about errors, dropped packets, overruns and frame are extracted as well. |
| rx_errors | Integer | |
| rx_dropped | Integer | |
| rx_overruns | Integer | |
| rx_frame | Integer | |
| tx_packets | Integer | The tx_* fields show the total number of packets transmitted. Additional information about errors, dropped packets, overruns and frame are extracted as well. |
| tx_errors | Integer | |
| tx_dropped | Integer | |
| tx_overruns | Integer | |
| tx_carrier | Integer | |
| rx_bytes | Integer | The total amount of data that has passed through the network interface (incoming). |

| | | |
|-------------------|---------|---|
| tx_bytes | Integer | The total amount of data that has passed through the network interface (outgoing). |
| broadcast | String | Denotes the broadcast address |
| inet | String | The device IP address |
| inet6 | String | The device IPv6 address |
| txqueuelen | Integer | The length of the transmit queue of the device. |
| mtu | String | The Maximum Transmission Unit is the size of each packet received by the network interface. |
| flags | Integer | The port number of the remote computer. An asterisk (*) is shown for the host if the server is listening on all interfaces. |

3.2 Devices running a Linux-/Windows-based OS distribution

For more powerful IoT devices equipped with a Linux- based distribution that allows access to the root OS, a Linux-based monitoring agent (software services) has been developed. A number of IoT devices bear lightweight Linux based distributions, and these devices are probably the ones providing most liberty in the monitoring and detection of various events. Such devices include embedded systems such as raspberry Pis, TV boxes etc. The developed service can be deployed on such devices and the integration with Device Profiling service and other Cyber-Trust components will be done.

For the implementation of the Linux-based Smart Device monitoring service, a software application has been developed that accesses system’s information, gather data and communicate them with Device Profiling Service.

3.2.1 Performance Monitoring

For Linux-Based devices as well as other OS like Windows, monitoring of RAM and CPU usage as well as tracking the storage capacity in regular intervals is done under the Cyber-Trust device tools. The following parameters are measured in regular intervals. Also, the aggregation of the values and preparation of averages over a specific number of seconds and minutes is available and is monitoring as well.

The monitored fields related with performance in Linux-based devices are the same with smart phone devices and are presented in Table 18. The metrics can be monitor in regular intervals with a variable period of time. The acquisition of performance monitoring metrics is done using, “psutil” package. The psutil package checks disk usage of the whole filesystem, RAM memory usage and CPU usage, the three of them as a percentage. Additional calculations can be applied in order to extract the full list of performance monitoring fields.

```
disk_usage = psutil.disk_usage('/').percent
cpu_load = psutil.cpu_percent(interval=SAMPLE_TIME)
mem_usage = getAverageMemoryUsage(SAMPLE_TIME, 1)

def getAverageMemoryUsage(sample_num, update_interval):
    mem_percentage = 0
    for i in range(0, sample_num):
        mem_percentage += psutil.virtual_memory().percent
        time.sleep(update_interval)
    return mem_percentage/sample_num
```

The above segment of code describes part of the implementation of SDM for Linux-/Windows-Based devices.

Similar with mobile app, the JSON messages are published on the Profiling Service where the current status, history and the information for the evidence DB is available. The information is correlated with the device under monitoring considering the device ID and the type of device given in the headers of the message.

3.2.2 Critical OS and Firmware Integrity Monitoring

For general purpose devices such as Linux/Windows-based hosted OS, we developed a Python script that is responsible for hash calculation, storage and comparison. A service has been developed based on the Python script where it accepts as an input a list with filesystem paths of the considered critical OS files where the hash function is applied. For the purposes of Cyber-Trust project and the calculation of the hash, the MD5 algorithm is used while different hashing algorithms and file integrity methodologies can be applied as well. Similar with the monitoring of performance metric, the calculation of hashes can be done periodic with variable time intervals. The calculation of a hash is followed with a comparison with the previous calculated hash of the considered file. All the hashes are posted and stored in the profiling service. The comparison between the consecutive calculated hashes can be done on the device itself (in the case that the hashes are already kept on the device storage) or on the profiling service and the data acquisition component. An alert is triggered in a case of a mismatch.

For calculating the hash, the algorithm used is MD5. Although it has been demonstrated to be weak to collision attacks (finding any two distinct files that have the same hash), MD5 is still useful for file integrity checking, since there are not any currently known practical or almost-practical second preimage attacks (modifying an input without changing the resulting hash, and therefore be able to replace files without hash changing) for MD5.

For each hash calculation, based on the input files/binaries list, the fields that presented in Table 19 are prepared and sent to the Profiling Service. About device details and list of installed applications is gathered from devices that is applicable.

Python based scripts have been prepared and deployed on Linux-/Windows-Based machines for the calculation of active processes as well as installed applications if it is applicable. Similar data messages as Table 20, Table 21 and Table 22 present, are collected from the device agents.

3.2.3 Device Level Network Monitoring

For Linux-based as well as Windows-based devices a Python script is developed in order to periodically monitor the activity related with ports. The operation of the script is mainly based on the usage of system-specific calls using *netstat* configured with the appropriate arguments and processing the output. The result of this script is again published to the profiling service. The comparisons can be executed either at device level or centrally at the profiling service environment.

```
aditess@pr_filestore: $ sudo netstat -ptnul
[sudo] password for aditess:
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      795/systemd-resolve
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1259/sshd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1983/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1967/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2018/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1977/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1964/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2009/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1993/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1980/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1965/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2011/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1032/rsync
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      910/memcached
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1982/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1966/python
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2014/python
tcp6       0      0 :::22                   :::*                    LISTEN      1259/sshd
udp        0      0 0.0.0.0:53              0.0.0.0:*               795/systemd-resolve
aditess@pr_filestore: $
```

Figure 10 Example of output from netstat -ptnul in Linux

Similar with other monitoring services, the period can be configured by varying the time interval of data collection. Alerts are generated based on the defined rules as well as when a port have been closed or opened.

3.3 Devices implemented to use IoT cloud Services

In the market, there are IoT devices that are design to integrate sensors and implement specific functionalities. As the computational budget of these devices is limited, the hosting of a lightweight operating system (e.g. Real-Time Operating System, RTOS) doing specific tasks is often. This kind of devices do not allow or may tolerate third-party services to be accommodated onboard the IoT device itself. Due to this, monitoring of such devices will be performed based on the exposed API of the IoT cloud network onto which it is built for communication with back-end cloud services. This integration approach, with the IoT service providers allows Cyber-Trust to enable monitoring of IoT devices that are not exposed any open interfaces and the acquisition of health and other monitoring data is done through cloud services or third-party interfaces.

For the purposes of the Cyber-Trust project, we investigated different IoT cloud service providers and identified the best solution is to proceed with Google Cloud IoT, as it is an already tested platform with the support of one of the biggest companies in the world. Moreover, the Google Cloud IoT platform allows the use of virtual IoT devices, and they provide client libraries in several languages to interact with the platform API and create these virtual devices. Also, they use MQTT protocol (or HTTP if needed) for communication. All of these features allow the smoothly integration with Cyber-Trust platform as well as the deployment of the large number of smart homes.

3.3.1 Create new device

For using the Google Cloud IoT platform and register devices, the first step is to create the device in the platform and grouped the device under the appropriate device registry offered by Google. Each device registry is configured with at least one Pub/Sub topic where devices can send telemetry data (using HTTP or MQTT protocols). Configuring more than one topic in a registry is useful in order to send different kinds of data to different topics.

Device registries and devices can be registered manually using the Google Cloud WebConsole or through an HTTP API which is also supported using client libraries (wrappers for the HTTP API) available for many

programming languages. This functionality allows the registration of devices and the integration with Cyber-Trust platform in an automated way the registration to google services is transparent to the user of Cyber-Trust.

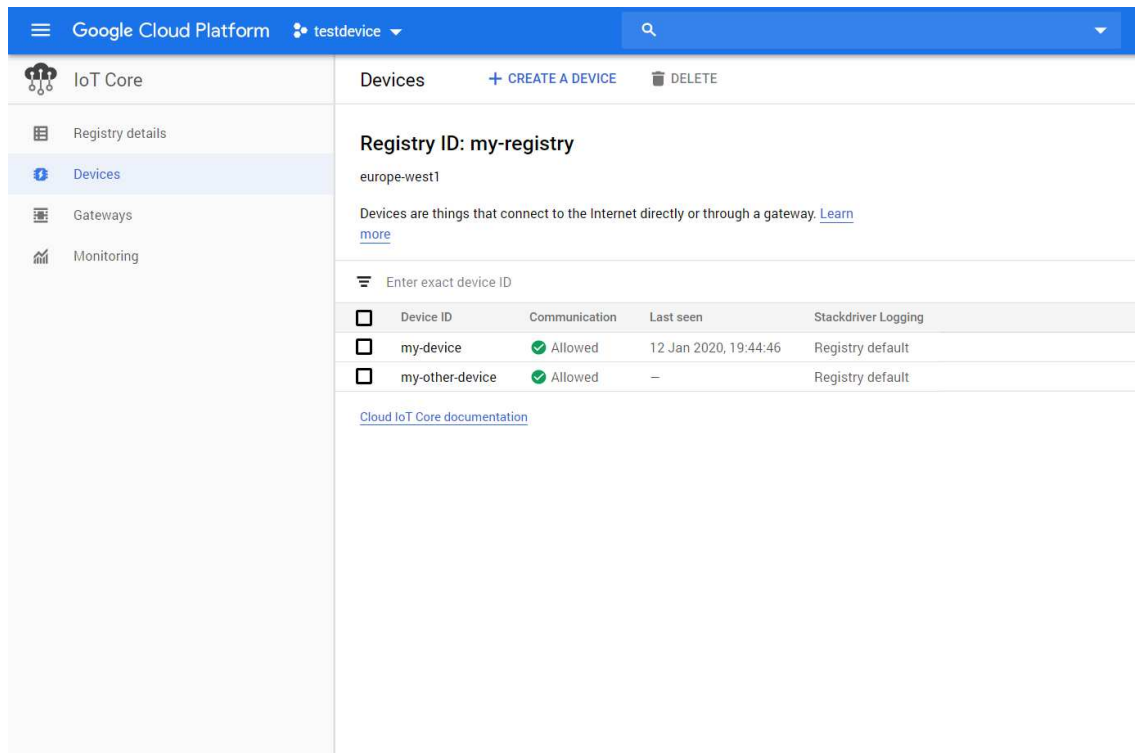


Figure 11 Google Cloud Platform IoT Web Console

3.3.2 IoT Cloud Services Data

Upon the registration of device to both platforms, Cyber-Trust and Google Cloud IoT, the monitoring of data is available. The Google service supports four different kind of data that can be exchange between the device and the IoT cloud service and therefore can reach the Cyber-Trust SDM related services. The four categories are listed below:

- **Device metadata:** During the creation and setup of a device, up to 500 key-value pairs can be specified. This class of data is static (fixed) and the update of data is happened sporadically.
- **Device state:** With a budget of 64KB data, the state of device can be synced with the cloud service periodically. The frequency of data transition from the device to the cloud service can be set up to 1 second. The type and format of data can be configured by the user.
- **Device configuration:** Similar with device state, this category of data can be configured by the user. The purpose of configuration data is to setup the device after the registration as well as to control the device during its operation. A device configuration logged is kept, so users can roll back to specific configuration at any given time. This is very important feature as allow the application of specific configuration after an event. Limitations about updating-frequency and size are the same as for device state.
- **Device telemetry:** Each device has a default pub/sub topic where it sends all telemetry information that can be later needed for the developer.

3.3.3 IoT Cloud Services Integration

Accessing the third-party cloud IoT services such as the Google Cloud IoT, all the authentication and authorization procedures provided by the service owner should be followed. This can be done using the API tokens and included them in the headers of the HTTP requests or simpler through the client libraries if there are available. In the case of Google Cloud IoT the authentication is done using the available python library (recommended by Google as well).

The first step in this process is the creation of the service account with the minimum permissions in order to allow access to the Google Cloud API.

In the following example, the credentials for the CyberTrust testing account have been passed through the parameters of the function in the format of a JSON object (w.r.t. `json_credent`).

```
def create_device_registry(json_credent, project_id, registry_id, cloud_region, pubsub_topic):
    client = iot_v1.DeviceManagerClient.from_service_account_json(json_credent)
    parent = client.location_path(project_id, cloud_region)

    body = {
        'event_notification_configs': [{
            'pubsub_topic_name': 'projects/{}/topics/{}'.format(project_id, pubsub_topic)
        }],
        'id': registry_id
    }

    response = client.create_device_registry(parent, body)

    return response
```

```
def create_device(credentials_json, project_id, registry_id, cloud_region, device_id,
certificate_file):

    client = iot_v1.DeviceManagerClient.from_service_account_json(credentials_json)

    parent = client.registry_path(project_id, cloud_region, registry_id)

    with io.open(certificate_file) as f:
        certificate = f.read()

    # Note: You can have multiple credentials associated with a device.
    device_template = {
        'id': device_id,
        'credentials': [{
            'public_key': {
                'format': 'RSA_X509_PEM',
                'key': certificate
            }
        }]
    }

    return client.create_device(parent, device_template)
```

To create a device, first of all it is necessary to create a public/private key pair for it. The public key is used for the Cloud IoT Core to authenticate each device.

Once the device resource is created in the Cloud Platform, it can be linked to a real device, ready to send data. When connecting IoT devices which are not built with any internet connection (Zigbee, Bluetooth devices, etc.) a gateway device can be configured.

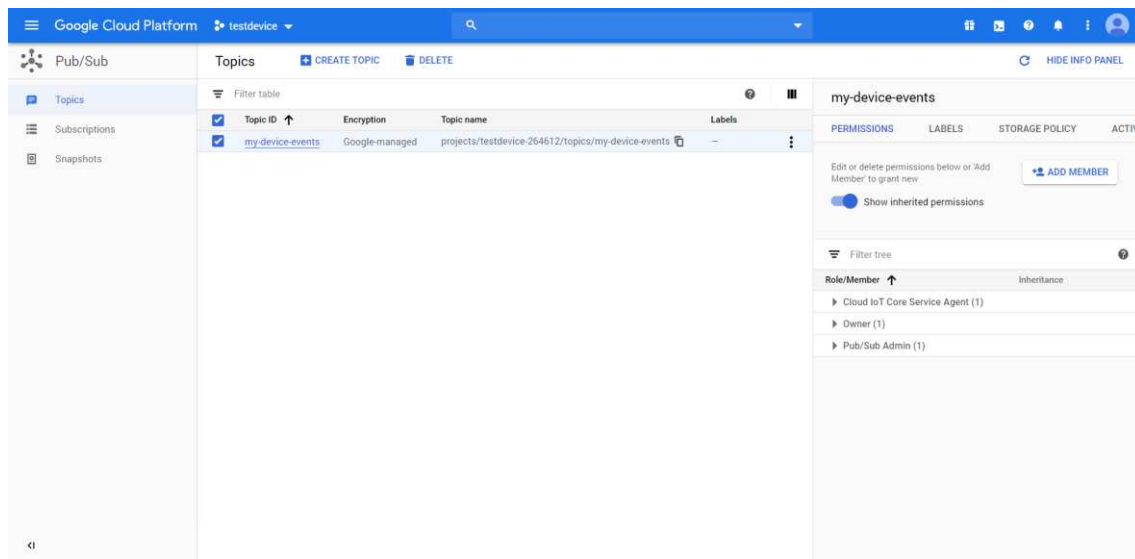


Figure 12 Pub/Sub Platform Webconsole

As we already mentioned, another important feature for Cyber-Trust is the emulation of virtual devices. This is also supported by the Google Cloud IoT where the emulated devices are sending HTTP or MQTT events to the Cloud API. The configuration of real-device regarding the pub/sub topics can be applied here as well. For authentication, devices must identify themselves by creating a JWT token (<https://jwt.io>) with their own private key and the exchanged data should be encoded in Base64 format.

```
def publish_telemetry(project_id, registry_id, cloud_region, device_id, private_key_file,
binary_data):
    endpoint = 'https://cloudiotdevice.googleapis.com/v1/projects/{}/locations/{} \
    /registries/{}/devices/{}:publishEvent'.format(project_id, cloud_region,
registry_id, device_id)

    jwt_token = create_jwt(project_id, private_key_file).decode()

    headers = {'authorization': 'Bearer {}'.format(jwt=jwt_token)}

    data = base64.b64encode(binary_data).decode()
    r = requests.post(endpoint, json={"binary_data": data}, headers=headers)

    return r
```

The above example implements a case where the data are sent to the Pub/Sub Platform, also integrated in Google Cloud, but different from IoT Platform. Therefore, Pub/Sub API must be used to retrieve telemetry data. For interacting with this API there is also a Python client library, that will be used in this example. A subscription (resource representing the stream of messages from a single, specific topic, to be delivered to the subscribing application) to the telemetry topic is created by default when creating the registry. However, more subscriptions are allowed to be created when needed.

```
def subscribe(credentials_json, project_id, subscription_name):
    subscriber = pubsub_v1.SubscriberClient.from_service_account_json(credentials_json)
    # The `subscription_path` method creates a fully qualified identifier
    # in the form `projects/{project_id}/subscriptions/{subscription_name}`
    subscription_path = subscriber.subscription_path(
        project_id, subscription_name)

    def callback(message):
        print('Received message: {}'.format(message))
```

```
message.ack()  
  
subscriber.subscribe(subscription_path, callback=callback)  
  
return subscriber
```

4. Device Forensic Evidence Collection

Apart from the storage of data related with the device profiles, vulnerabilities and software updates, Profiling Service is the responsible component for host the Evidence DB database. Considering the features of Profiling Service and core DMS (Section 2) related with privacy and security aspects, such as data integrity, audit, secure communication and data encryption, a dedicated storage space both for binaries as well as metadata is prepared for the Evidence DB.

Despite the fact that Evidence DB is hosted under the Profiling Service, the Evidence DB data are totally isolated from the rest of data and no direct interaction is allowed. Profiling Service can communicate to Evidence DB through the available REST API.

Two main data formats are kept in the Evidence DB:

- **Binary Files:** Information of device data in a binary format. The extraction of data, that include the monitoring logs, is done from the profiling service and encoded in base64 format.
- **Metadata:** For each binary file, a set of metadata is prepared and stored in the metadata database of the Evidence DB. The fields of the metadata are presented in Table 25.

Table 25 Evidence DB Metadata Fields

| Field | Type | Description |
|-------------------------|-----------------------------------|---|
| type | String (ENUM) | The type of the evidence records. i.e. Application, OS, etc. |
| name | String | Name of the evidence record |
| description | String | Description of the evidence that is recorded |
| date | String | Date of the event occurrence that triggered evidence storage. |
| url | String | The URL of the source of the event that caused evidence storage. |
| source | String | The Component that generate the event that caused evidence storage. |
| vulnerability | String (Profiling Service Ref ID) | Reference to vulnerability related with the evidence record if exists. |
| attack | String (Profiling Service Ref ID) | Reference to the attack related with the evidence |
| metatada | JSON Object | Metadata info related with the type of evidence record |
| metadata.name | String | The name of software or OS that is responsible for the evidence record |
| metadata.version | String | The version of the software or OS that is responsible for the evidence record |
| metadata.type | String | The type of the evidence records. For example, if it's application the type of the application. |

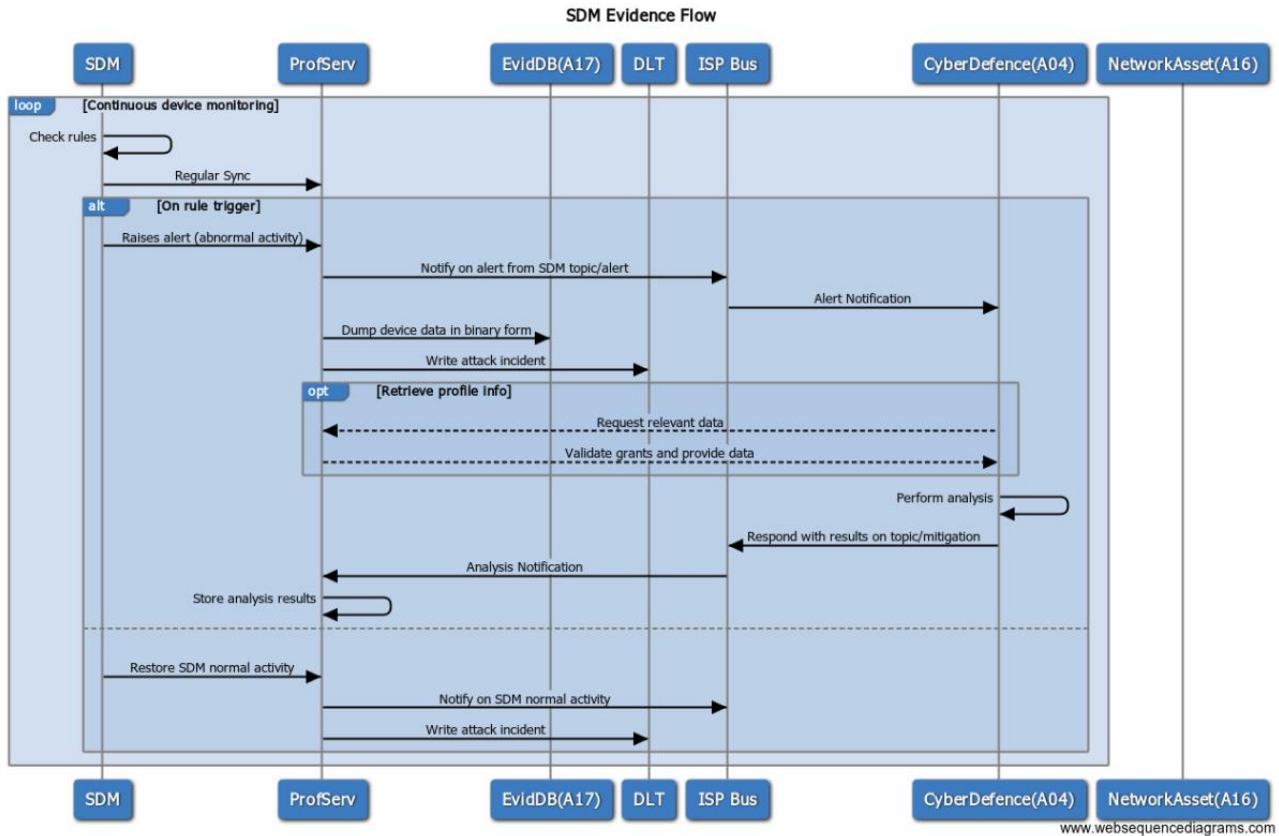


Figure 13 Data Flow for Evidence DB

Figure 13 presents that data flow and the interaction between Cyber-Trust component for the Evidence DB. The profiling service is the only component that is directly communicating to post data to the Evidence DB. When an alert is raised and the Cyber-Trust components are informed through the Information BUS, the Profiling Service will be able to create dump and upload data to the Evidence DB. In particular, the steps below are followed by the profiling service:

1. Profiling Service is requested to extract evidence for the Evidence DB.
2. Using the monitoring end-points with the appropriate parameters a request of data is placed.
 - a. Selection of monitoring category(ies)
 - b. Define time window for data query
 - c. Retrieve the monitoring data
3. The JSON is converted to binary using base64 encoding
4. Binary signature is calculated (MD5 or similar algorithm)
5. POST data to the evidence DB
6. Confirmation response

5. Conclusion

This deliverable performed a detailed documentation of monitoring activity for IoT devices under the Cyber-Trust project. The main achievements of the work under this report are presented below:

- Design and Development of Device Profiling Service: This action include the definition of data models that are hosted under the profiling service and mainly are responsible for the acquisition of data, rules application and alerting mechanism part of the device monitoring process.
- Development of a comparator service that can be hosted either at device layer or centralized for the application of rules towards the detection of abnormal activity.
- Data correlation of vulnerabilities and patches with devices based on the calculation of CPE number.
- Detailed description and implementation of monitoring agents targeting three types of activities: (a) performance monitoring, (b) critical files and firmware integrity through signatures and hash functions and, (c) network related data.
- Development of a mobile app as a Cyber-Trust UI tool for users that will be able to retrieve information about the device activity as well as notifications related with Cyber-Trust platform.
- Introduction of Evidence DB for the extraction of monitored information to be stored in a secured and privacy by design environment.

The next steps of this work are the detection of attacks and the application of mitigation actions on devices. The outcome will be reported in the second deliverable of Task 6.2, D6.6 “Device-level attacks: proposed solutions”.